ISO-KTSP

Trabajo Final de Grado

Grado en Ingeniería en Informática

Especialidad: Computación

Michał Zawisza Alvarez

Director: Eduardo Eyras

Ponente: Xavier Messaguer (dep. CS)

Enero de 2020

ÏNDICE

1. INTRODUCCIÓN Y CONTEXTUALIZACIÓN

El presente documento es la memoria del Trabajo de Final de Grado (TFG) titulado iso-kTSP (de "isoform k top scoring pairs") del Grado en Ingeniería Informática de la Universitat Politècnica de Catalunya. La parte práctica del proyecto se realizó entre marzo de 2013 y junio de 2014, como colaborador externo del Institut Hospital del Mar d'Investigacions Científiques (IMIM), sin retribución económica. El proyecto consistió en implementar un algoritmo de machine learning existente, el k-TSP (k top scoring pairs), añadiendo una variación nueva para cubrir las necesidades específicas de una investigación en curso en genómica del cáncer. El proyecto se realizó integrado en la investigación del Biología Computacional del ARN del Programa de Informática Biomédica (IMIM y Uniiversitat Pompeu Fabra, UPF), con el Prof. Eduardo Eyras como tutor del IMIM y el Prof. Xavier Messeguer como tutor de la UPC, y se enmarcó en el proyecto de investigación del Dr. Endre Sebestyén y el Prof. Eduardo Eyras. El resultado fue el software iso-kTSP, que se usó y se incluyó como parte del artículo científico publicado en Nucleic Acids Research.

1.1 MOTIVACIÓN Y OBJETIVOS CIENTÍFICOS

Este TFG se enmarca en el contexto de la investigación en genómica regulatoria del cáncer, y en concreto en un mecanismo genético concreto, el *splicing* alternativo. Esto hace que para comprender la motivación y los objetivos científicos que dan lugar a este TFG, se requieren unos conocimientos específicos en biología celular, genética y biología del cáncer que no suelen tener personas no especializadas. Debido a la cantidad de información que esto implicaría explicar, se ha optado por hacer un documento anexo, divulgativo y sin tecnicismos innecesarios, para facilitar el entendimiento a nivel orientativo de los conceptos biológicos necesarios, desde la base hasta los temas más específicos, para comprender la nomenclatura y el razonamiento detrás de este trabajo. Es recomendable leer el anexo antes de proceder con la lectura del resto de este documento. De la explicación dada en el documento anexo, cabe destacar los siguientes puntos, que nos llevan de manera directa al planteamiento del proyecto de investigación y a las preguntas que pretende responder este trabajo:

- El *splicing* es uno de los muchos mecanismos que intervienen de manera habitual en el proceso de expresión de genes en las células.
- El *splicing* alternativo permite a las células expresar distintas variantes del mismo gen, llamadas isoformas.
- La regulación del *splicing* alternativo permite que las distintas isoformas se expresen donde y cuando se necesitan. Es un proceso complejo y aun requiere estudio, pero se sabe que determinados puntos de la secuencia del gen expresado son importantes para esa regulación (además de que indirectamente estén implicadas muchas otras secuencias del genoma).

- El cáncer es una enfermedad, en muchos casos grave, que se desarrolla cuando una célula acumula mutaciones en puntos clave de su genoma, de manera que, por un lado, deja de responder a los mecanismos de seguridad del cuerpo para la eliminación de células defectuosas y, por otro lado, empieza a proliferar descontroladamente, afectando al funcionamiento de las células sanas de su alrededor y del cuerpo en general.
- El cáncer es una enfermedad compleja y difícil de tratar, debido en gran parte a que las células que causan la enfermedad son células del propio organismo, lo cual dificulta encontrar fármacos que las eliminen o neutralicen sin afectar a las células sanas. Entender el mecanismo molecular que lleva al cáncer y sus puntos clave podrían ayudar a crear maneras de combatirlo más eficazmente.

De estos puntos surge el planteamiento básico de la línea de investigación que desarrollaron Endre Sebestyén y Eduado Eyras: si en las células cancerígenas hubiese mutaciones en sitios del genoma que son importantes para la regulación del *splicing* alternativo, podrían detectarse diferencias en la expresión de isoformas de algunos genes respecto al funcionamiento normal de las células. Si además se detectasen cambios en la expresión de isoformas de determinados genes que fuesen consistentes en determinados tipos de cáncer, querría decir que esos cambios forman parte del proceso principal del desarrollo celular de ese tipo de cáncer, es decir, estos serían marcadores de *splicing* alternativo para ese tipo de cáncer.

Encontrar estos marcadores no necesariamente querría decir que esos cambios concretos de regulación del *splicing* alternativo sean la causa del cáncer (podrían ser síntomas celulares secundarios), pero sí daría pistas para centrar la investigación en esos genes y poder llegar a una mejor comprensión de la enfermedad. Además, para determinados tipos de cáncer en los que es habitual hacer biopsias para controlar y detectar el cáncer, estos marcadores podrían ser una herramienta adicional de diagnóstico.

1.2 EL PUNTO DE PARTIDA: LOS DATOS DE EXPRESIÓN DE GENES E ISOFORMAS

Para poder analizar cambios de expresión de las distintas isoformas hace falta cuantificar dicha expresión. La técnica más adecuada para hacerlo se llama RNA-seq, y consiste en extraer una muestra del tejido de interés, purificar el RNA y secuenciarlo. Cuanta más señal de secuenciación haya en los exones específicos de cada isoforma, mayor será la expresión de esa isoforma respecto a las otras isoformas del mismo gen.

Como queremos ver las diferencias entre tejidos sanos y tejidos con cáncer, necesitamos obtener muestras de ambos tejidos. En el caso de los datos con los que hemos trabajado en nuestro proyecto, los datos se obtienen de experimentos previos: los pacientes dan su consentimiento para que sus datos se usen con fines científicos y las biopsias de los

tejidos de interés se secuencian mediante RNA-seq. Los datos obtenidos se publican en bases de datos para uso por toda la comunidad científica.

Así pues, el proyecto de investigación parte de la cuantificación de la expresión de genes e isoformas, para todos los genes codificantes presentes en el genoma humano (unos 20000), organizados en sets para distintos tipos de cáncer con varias decenas de muestras de tejido afectado por la enfermedad y varias decenas de muestras de tejido control no afectado. La cantidad de datos y la característica del objetivo de investigación hacen que el *machine learning* sea la herramienta idónea para este proyecto de investigación.

2. ESTADO DEL ARTE

2.1 *MACHINE LEARNING* EN GENÉTICA Y GENÓMICA

La capacidad de hacer predicciones y definir comportamientos sobre variables basada en los datos observados, que es la esencia del *machine learning*, lleva aplicándose, en la ciencia en general y en la biología molecular en particular, desde el siglo XIX con el desarrollo de la estadística. El *machine learning* y el *data mining* son una evolución de los métodos estadísticos, que son capaces de mejorar y refinar los modelos predictivos que salen del análisis de los datos teniendo en cuenta los aciertos y fallos que tiene el modelo predictivo. Ya en 1951, Minsky y Edmonds implementaron la primera red neuronal, estrategia que sigue siendo la base de muchos algoritmos de *machine learning*. Desde entonces, una gran variedad de estrategias y algoritmos cada vez más potentes han contribuido, junto con la enorme cantidad de datos disponibles (gracias, entre otras cosas a la ubiquidad del uso de Internet) a que hoy en día el *machine learning* tenga una importancia fundamental en distintas áreas, desde las estrategias de negocios en empresas hasta la predicción del tiempo [1].

En el campo de la biología y de la genética, ya en los años 90 surgen los primeros usos del *machine learning* en bioinformática. Pierre Baldi [2] ya en 1998 centraba su libro en distintas estrategias (principalmente redes neuronales y modelos ocultos de Markov) y aplicaciones de *machine learning* en genética y proteómica. En las dos primeras décadas del siglo XXI, los avances constantes en las técnicas de secuenciación que permiten secuenciar genomas enteros con facilidad y a un coste relativamente bajo han causado una explosión en la cantidad de datos disponibles. Eso a su vez ha aumentado aún más la importancia de las estrategias de *machine learning* para analizar grandes cantidades de datos, incluidos genomas enteros de miles de millones de pares de bases.

La variedad de aplicaciones de *machine learning* en bioinformática ha ido también en aumento, desde los campos tradicionales, como la obtención de estructuras de proteínas o la anotación de genes y la secuenciación con *microarrays*, hasta los llamados *Genome-wide Association Studies* (GWAS) o la detección por imagen de tumores. La variedad de estrategias de *machine learning* usadas hoy en día en esta gran diversidad de aplicaciones bioinformáticas incluye algoritmos *Naive Bayes*, clasificadores lineales, algoritmos basados en árboles, clasificadores basados en distancia algoritmos *ensemble* y algoritmos no lineales basados en *kernels* [3] [4].

2.2 EL ALGORITMO K-TSP

Pasemos ahora a analizar más en detalle las soluciones al problema científico concreto que nos ocupa: detectar los genes que cambian de forma consistente la expresión de sus isoformas cuando se comparan datos de células sanas con células afectadas por cáncer. En el momento de empezar a trabajar en este proyecto, en el año 2013, a nuestro saber no existía una solución concreta de *machine learning* que permitiese esta

tarea diferenciando las isoformas de un mismo gen. Sí que existían, sin embargo, diferentes algoritmos de *machine learning* que permitían realizar este análisis con los datos de expresión totales de los genes.

En un artículo de 2005, Tan *et al*. [5] proponen el algoritmo k-TSP para la clasificación de datos de tejidos con y sin cáncer en función de los cambios de expresión de genes y los comparan con otros algoritmos que tienen una elevada fiabilidad en la predicción, como por ejemplo el análisis predictivo de *microarrays* o el *support vector machine*, entre otros. Argumentan que los algoritmos de *machine learning* usados hasta entonces, aunque fiables en la predicción, acaban produciendo normas predictivas complejas y de difícil interpretación, haciendo difícil la obtención de pistas sobre el mecanismo biológico subyacente. En los tests, el algoritmo k-TSP resulta tener una fiabilidad comparable a los mejores algoritmos con los que se compara, pero generando reglas sencillas, que involucran unos pocos genes, con lo cual hacen más viables estudios posteriores sobre los posibles mecanismos biológicos, centrándolos en los genes que aparecen en el modelo predictivo del algoritmo.

El algoritmo es una variante del algoritmo TSP (*top scoring pairs*). Se basa en transformar los valores absolutos de expresión de cada gen en un valor de ranking para cada muestra (del gen menos expresado al más expresado dentro de cada muestra). Después, compara dos a dos todos los pares de genes y mira la diferencia de ranking entre las muestras de cáncer y las sanas. El algoritmo TSP escoge la pareja de genes que más consistentemente ha hecho una inversión en el ranking. El algoritmo k-TSP construye su modelo a partir de las k parejas de genes que tienen inversiones en el ranking de manera más consistente. El hecho de usar la posición en el ranking en lugar del valor absoluto de expresión ayuda, entre otras cosas, a dar robustez frente a cambios naturales en la expresión de todos los genes y también frente a la variabilidad técnica de los experimentos de los que se obtienen los datos.

3. OBJETIVO Y ALCANCE

Antes de definir los objetivos, es conveniente pensar en los *stakeholders*, las personas o entidades que podrían verse afectados directa o indirectamente por este proyecto, para poder tener en cuenta si los intereses de los *stakeholders* se ven reflejados en los objetivos y metodología que se definen. De mayor a menor importancia:

- El entonces investigador postdoctoral Endre Sebestyén y el investigador principal del grupo, Eduardo Eyras: este proyecto se enmarca en investigación de Endre Sebestyén, dirigida por el Prof. Eduardo Eyras y pretende el desarrollo de las herramientas de software que permitan obtener los resultados científicos necesarios para avanzar en el proyecto de investigación.
- El Institut Hospital del Mar d'Investigacions Científiques (IMIM): es la entidad de la que forma parte el grupo de investigación y es la que proporciona el espacio de trabajo y la infraestructura necesaria para el desarrollo del proyecto (PC con un entorno de trabajo adecuado, los servidores del grupo de investigación para hacer los tests y ejecuciones finales, etc.).
- La Universitat Politècnica de Catalunya (UPC): este proyecto se enmarca como TFG de un alumno de la facultad de Informática de la UPC, por lo tanto, el desarrollo y el resultado del proyecto pueden tener un impacto en la imagen y reputación de la UPC.
- El resto de la comunidad científica, en especial la que se dedique a la genética del cáncer: por un lado, los resultados científicos obtenidos con la ayuda del software desarrollado en este proyecto podrían ser de relevancia para posteriores estudios científicos. Por otro lado, aunque este software está especialmente dirigido a las necesidades de la investigación del Dr. Endre Sebestyén y el Prof. Eduardo Eyras, podría haber otros proyectos de investigación interesados en usar el software en contextos parecidos.
- Las personas de las que se han obtenido los datos de expresión génica: aunque el software no presenta la perspectiva de la publicación directa de los datos de entrada, debido a que no se prevé la necesidad de tal funcionalidad, el proyecto no debería comprometer de manera razonable la privacidad de los datos con los que trabaja.
- Futuros pacientes de cáncer: de manera muy indirecta, los avances científicos que se puedan obtener a partir del uso de este software podrían dar lugar a avances médicos en la prevención. detección y tratamiento del cáncer.

Así, el objetivo general del proyecto es implementar una herramienta bioinformática que utilice estrategias de *machine learning* para clasificar entre datos de expresión de isoformas de células con cáncer y células sanas y dar el modelo predictivo que pueda contener información sobre los cambios de expresión posiblemente significativos y qué isoformas de que genes presentan esos cambios.

Para dar respuesta al objetivo general, se plantea hacer una implementación propia del algoritmo k-TSP, con la opción de poder trabajar sobre los datos de expresión de isoformas, además de los datos de expresión de genes. La modificación más importante en el funcionamiento del algoritmo en su versión para isoformas es que los pares de isoformas que se tienen en cuenta en el modelo predictivo solo pueden ser de dos isoformas del mismo gen, de manera que se pongan de manifiesto los cambios en la regulación del *splicing* alternativo.

Antes de hablar del alcance, es necesario hacer una consideración: los conocimientos en programación en el ámbito de la bioinformática son muy variables, y es de esperar que la mayor parte de los potenciales usuarios de este software no tendrán las habilidades para hacer tareas de refinación y mantenimiento del código. Así, es una prioridad que las funcionalidades sean suficientes para cumplir los objetivos, pero las funcionalidades secundarias o adicionales son menos prioritarias que la robustez del software.

Para definir el alcance, definimos las funcionalidades principales que tendrá que tener el software y las secundarias o complementarias, que son deseables, pero no imprescindibles:

Funcionalidades principales:

- Ejecutar el algoritmo k-TSP básico para datos de transcripción de genes.
- Ejecutar el algoritmo modificado para que trabaje con los datos de transcripción de las isoformas.
- Presentar los resultados, tanto el modelo en si como la fiabilidad de la predicción

Funcionalidades secundarias o complementarias:

- Posibilidad de hacer la predicción con un modelo de entrada, ya sea creado por el usuario u obtenido de una ejecución anterior.
- Opciones adicionales de personalización de la ejecución y presentación de resultados, con información adicional sobre la ejecución
- Control de errores de ejecución con la máxima información posible en los mensajes de error.
- Integración con algún sistema de visualización gráfica de los resultados.

Se deja fuera del alcance del proyecto la posibilidad de incluir soporte para distintos formatos de entrada y de salida. Los detalles del formato se dan en el documento de especificación técnica.

4. METODOLOGÍA Y DECISIONES TÉCNICAS INICIALES

En primer lugar, comentar que, en el momento de empezar el proyecto, se barajaban dos opciones para el lenguaje de programación en el que se implementaría el proyecto: C++ o Java. Estas dos opciones eran las más familiares y en las que tenía acumulada más experiencia. El gran volumen de estudio preparatorio en biología y genética hacía que aprender un lenguaje de programación específicamente para el proyecto no fuese una opción. Entre las dos opciones, analizamos las ventajas e inconvenientes:

- C++: potencialmente más eficiente en cuanto a tiempo y uso de memoria. Mucha familiaridad con el lenguaje en sí, pero poca experiencia con el uso de entornos de desarrollo (IDE) con C++. Conocimiento del lenguaje menos extendido en el ámbito de la bioinformática y por lo tanto, menos facilidades para modificaciones y ampliaciones por usuarios interesados en ello.
- Java: menor control de la eficiencia y del uso de memoria. Mucha familiaridad tanto con el lenguaje como con el IDE NetBeans. Conocimiento del lenguaje ligeramente más extendido dentro del ámbito de la bioinformática.

La decisión fue la de usar Java: la familiaridad con NetBeans fue el factor más decisivo, y la pequeña perdida de eficiencia es asumible, si además el código final es más entendible. Además, entre las personas que se dedican a la bioinformática ni C++ ni Java son lenguajes muy extendidos (los más conocidos son lenguajes de scripting, como Perl o Python). Sin embargo, Java tiene fama de ser algo más accesible que C++, y por lo tanto es esperable que un mayor número de potenciales usuarios tengan conocimientos de Java, cosa que es interesante para posibles modificaciones o ampliaciones.

En cuanto a las decisiones de diseño que marcarán la metodología adaptada, hay que tener en cuenta que este proyecto no requiere un trabajo importante en cuanto a diseño de software, ya que no presenta una complejidad en la conceptualización del problema y de los datos que maneja (se trata de un algoritmo que hace comparaciones dos a dos entre datos de una matriz bidimensional). Esto resultará en un diseño de clases bastante simple. Aun así, se usará una arquitectura por capas, con clases específicas para leer y escribir ficheros, separadas de la lógica del algoritmo, de manera que sea relativamente fácil añadir o modificar la información de salida que se obtiene del programa.

Así, la metodología de trabajo será bastante simple: una implementación lineal de las funcionalidades, según el orden lógico de ejecución, y cada unidad funcional importante (función, clase o incluso fragmento de código importante), se testeará incrementalmente: antes de pasar a la siguiente unidad funcional, se asegurará dentro de lo razonable, que la unidad funcional que se ha acabado de implementar funciona correctamente y de manera robusta, integrada en el resto del sistema ya implementado. Debido a la cantidad, relativamente alta de datos y a la complejidad computacional, se

valorará la eficiencia de la implementación de ciertas unidades funcionales, y si no es satisfactoria, se pasará a valorar posibles mejoras.

Así, para cada unidad funcional importante, el esquema metodológico (fig. 1.) será el siguiente:

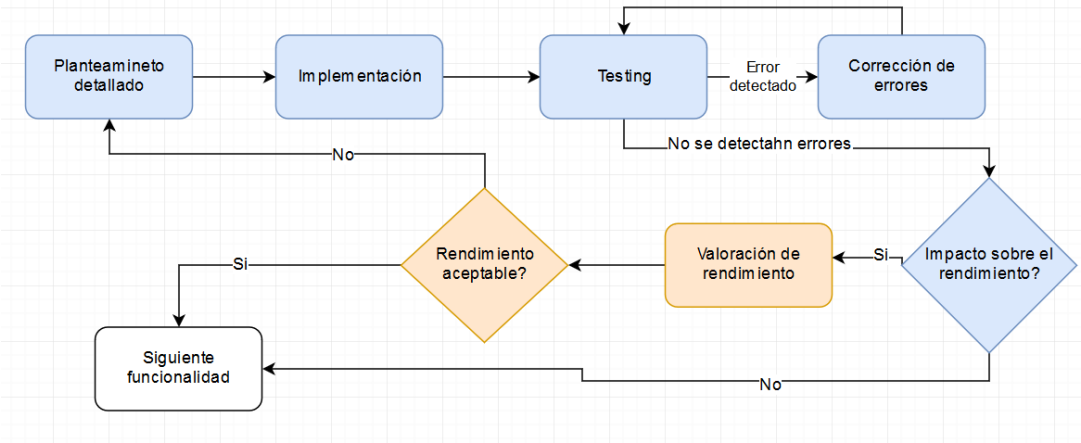

Fig. 1. Esquema de la metodología general para cada funcionalidad. *Elaboración propia*

Los tests funcionales se adaptarán a la funcionalidad que se esté implementando, pero generalmente se harán en primer lugar tests con un pequeño subconjunto de los datos de entrada completos, para después hacer tests con uno de los ficheros de datos completo. Si se considera oportuno, se harán pruebas con distintos ficheros de datos. En caso de que la funcionalidad tenga un coste computacional importante, se procederá a hacer tests de rendimiento, siempre con los ficheros de datos completos.

Además, el contacto, el tutor del proyecto, el Prof. Eduardo Eyras, será muy frecuente, con reuniones semanales del grupo de investigación y una disponibilidad muy alta y fácil para consultas en otros momentos. Eso facilitará resolver dudas, tanto sobre el contexto biológico como sobre el funcionamiento deseado del algoritmo, y ya en la última fase, poder consultar y discutir las mejoras y ampliaciones.

5. PLANIFICACIÓN

A continuación, nos centraremos en la organización y la planificación del proyecto. Debido al tiempo que ha transcurrido entre la realización del proyecto y la realización de esta documentación, y que en su día no existía una planificación formal, esta planificación es una aproximación de cómo se habría planteado la planificación al iniciarse el proyecto.

Además, como se explica más adelante, la duración real del proyecto, de casi un año y medio, responde tanto a algún retraso o replanteamiento de la planificación, como a la dedicación semanal limitada, al estar trabajando a media jornada en una empresa.

5.1 DELIMITACIÓN Y DESCRIPCIÓN DE LAS TAREAS

Las tareas se presentan en tres grupos: formación; diseño, implementación y testeo, y documentación y gestión. Hay que tener en cuenta que en las tareas de implementación y testeo se intenta dar un margen razonable para el testeo y la corrección de errores, pero sigue siendo una estimación que puede ser superada según las circunstancias.

Código tarea	Nombre	Previsión (horas)	Descripción
FB	Formación básica en biología	40	Formación en los conceptos básicos de biología a nivel de bachillerato, a través de lectura autónoma de libros de texto de bachillerato.
FE1	Formación esencial en genómica	50	Formación en los conceptos más esenciales y específicos para comprender los conceptos genéticos más esenciales para el proyecto, como la regulación génica, y el splicing alternativo a través de la lectura autónoma de bibliografía y consultas con el Prof. Eduardo Eyras
FE2	Formación esencial en el algoritmo kTSP	10	Comprensión detallada del funcionamiento del algoritmo kTSP, a través de la lectura detenida del artículo en el que se describe el algoritmo y consultas con el Prof. Eduardo Eyras
FC1	Formación complementaria en genómica general y genómica del cáncer	70	Formación complementaria para conocer el marco más general en el que se engloba el proyecto de investigación, a través de la lectura autónoma de bibliografía
FC2	Formación complementaria	20	Formación complementaria para conocer las bases y los principios generales del Machine learning

	en Machine learning		
FC3	Formación complementaria en reuniones del grupo de investigación	70	Formación complementaria para conocer los progresos de los distintos miembros del grupo de investigación en sus respectivos proyectos, con reuniones y presentaciones semanales de entre hora y hora y media
ID	Fase de primeras decisiones de diseño	5	Analizar y decidir los aspectos técnicos y de diseño del software de la implementación
I1	Implementación y testeo Fase 1	10	Implementación y testeo de las clases y funciones necesarias para leer los datos de entrada
I2	Implementación y testeo Fase 2	40	Implementación y testeo del cálculo de la matriz de ranking
I3	Implementación y testeo Fase 3	40	Implementación y testeo de las comparaciones dos a dos en la matriz de ranking y elaboración del modelo predictivo final
I4	Implementación y testeo Fase 4	20	Implementación y testeo de la fase de testing con el modelo creado por el algoritmo
I5	Implementación y testeo Fase 5	25	Implementación y testeo de la versión para isoformas del algoritmo kTSP
I6	Implementación y testeo Fase 6	30	Implementación y testeo de las clases y funciones necesarias para la presentación básica de los resultados y de los mensajes de error
IT	Testeo y debugging	15	Tests finales sobre el software básico completo y posible debugging de errores
IC1	Implementación y testeo complementario 1	15	Implementación y testeo de funcionalidades y opciones de salida complementarias
IC2	Implementación y testeo complementario 2	25	Implementación y testeo de la integración con un sistema de generación de gráficos
GP	Gestiones administrativas	5	Diversas gestiones relacionadas con el proyecto, como la firma del convenio y la acreditación en el IMIM
GD	Documentación	60	Documentación y redacción de la memoria del proyecto

Tabla 1: División de tareas. *Elaboración propia*

5.2 DIAGRAMA DE GANTT

El diagrama de Gantt refleja la planificación inicial del proyecto. Por compromisos laborales, la dedicación diaria se estimaba a 3 horas, y en el diagrama de Gantt se aproxima la duración en días de cada tarea teniendo en cuenta esa dedicación diaria, a excepción de la actividad FC3, que se ha hecho la estimación contando que medio día de dedicación equivale a 2 horas, que era el tiempo máximo que duraban las reuniones del grupo de investigación.

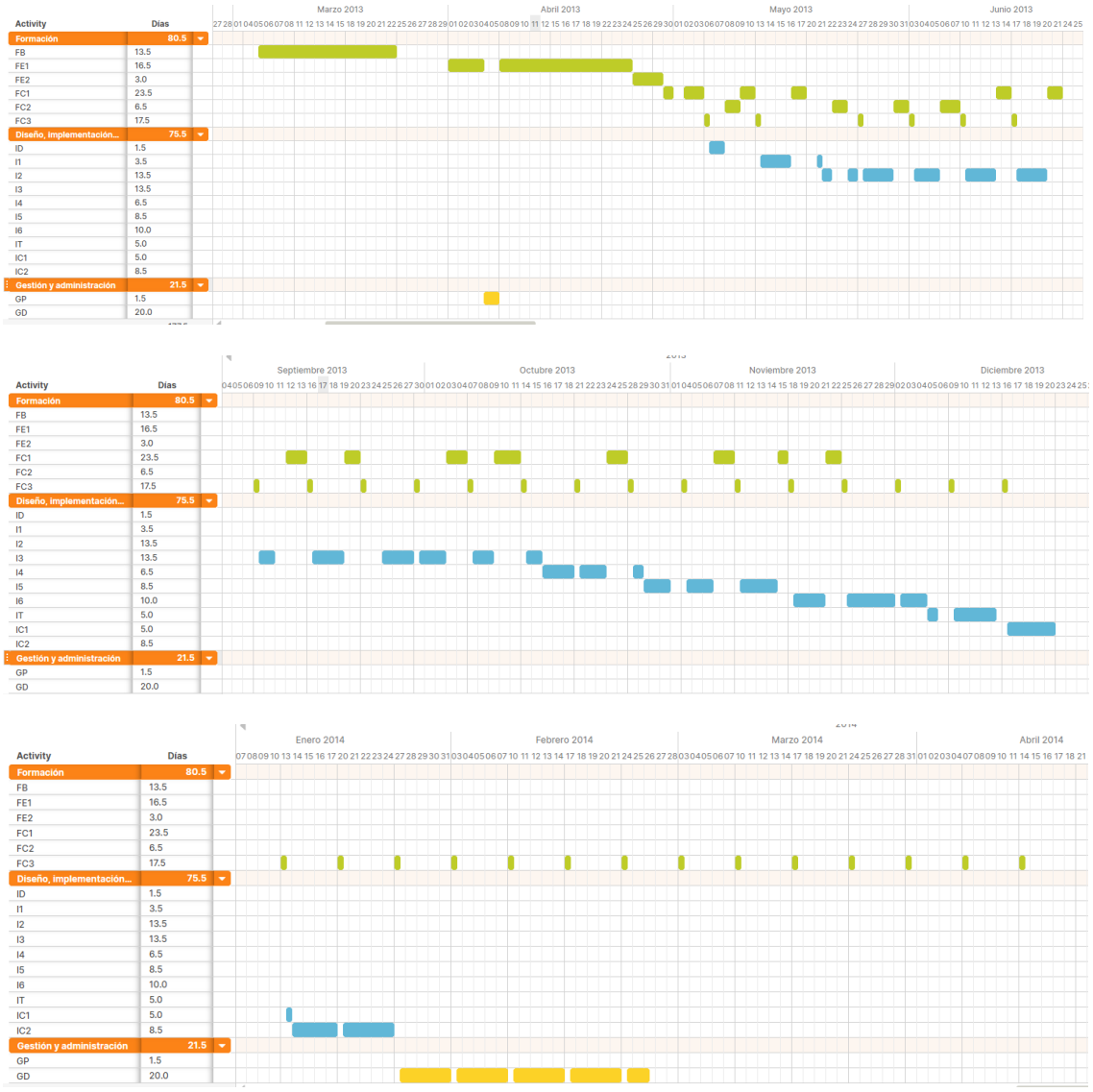

Fig. 2. Diagrama inicial de la planificación del proyecto. Elaboración propia.

5.3 GESTIÓN DE RIESGOS

Los riesgos principales que se prevén tienen que ver con retrasos en la implementación:

- Dificultades de implementación: puede haber detalles de la implementación que requieran soluciones a priori no evidentes, o que requieran más tiempo del previsto.

- Testeo y corrección de las unidades funcionales más largo de lo esperado: aun cuando se incluye un margen en la previsión de horas de cada tarea de implementación que se cree razonable para el testeo exhaustivo y la depuración de los errores de ejecución, cabe la posibilidad de encontrarse con un numero de errores mayor del previsto.
- Agrupando los dos puntos anteriores conjuntamente, se podría estimar que, en tareas individuales, en el peor de los casos podría llegar a aumentar hasta un 30% la dedicación real respecto a la prevista. Sin embargo, debido a que no se deberían dar estos retrasos en todas las tareas, y que algunas tareas podrían llegar a finalizarse con menos dedicación de la prevista (ya que ya se incluye cierto margen dentro de la previsión), en el peor de los casos esperaríamos un 15% de desviación en la dedicación global del proyecto por estas dos razones.
- Replanteamiento y reimplementación completa o parcial de una unidad funcional importante: se puede dar el caso que una implementación inicial de una parte del software no sea satisfactoria, sobre todo por cuestiones de rendimiento temporal, y se tome la decisión de replantear y volver a implementar o hacer cambios substanciales a esa parte del código para mejorarla.

Los dos primeros riesgos expuestos anteriormente, si no ocurren con una severidad o frecuencia muy importantes, deberían compensarse con el margen que se ha previsto en las otras tareas de implementación, con lo cual no deberían tener un impacto muy importante. En concreto, agrupando estos dos riesgos, se podría estimar que, en tareas individuales, en el peor de los casos podría llegar a aumentar hasta un 30% la dedicación real respecto a la prevista. Sin embargo, debido a que no se deberían dar estos retrasos en todas las tareas, y que algunas tareas podrían llegar a finalizarse con menos dedicación de la prevista (ya que ya se incluye cierto margen dentro de la previsión), en el peor de los casos esperaríamos un 10% de desviación en la dedicación global del proyecto por estas dos razones.

El último de los riesgos expuestos, sin embargo, si podría afectar de manera importante a la planificación, y creemos que daría lugar a una tarea extraordinaria. La estimación de la duración de esta tarea extraordinaria dependerá en gran medida de la complejidad de implementar la nueva solución respecto a la solución original, pero podría estar entre un 70% i un 150% de la previsión de la tarea original. Aunque todas las tareas de implementación son susceptibles de replantearse, las tareas I2 e I3 son las que corresponden a funcionalidades que podrían tener un mayor impacto en el rendimiento temporal, y por lo tanto, son estas tareas las que tienen mayor probabilidad de replantearse.

En cuanto a las decisiones que se podrían tomar en relación al control de esos riesgos, se presentan dos opciones:

- Una prolongación del proyecto. Cabe la posibilidad de alargar el proyecto más allá de lo previsto en la planificación, siempre y cuando esté disponible en un tiempo razonable para su uso en la investigación del Dr. Endre Sebestyén.
- Cancelación o acotación de alguna de las funcionalidades complementarias incluidas en las tareas IC1 o IC2: debe priorizarse la compleción de las funcionalidades básicas del proyecto, con una robustez suficiente para permitir un mantenimiento mínimo o nulo del código. Por lo tanto, reducir o cancelar las funcionalidades complementarias no debería considerarse como crítico para el éxito del proyecto.

5.4 MODIFICACIONES A LA PLANIFICACIÓN ORIGINAL

Aun cuando se ha hecho una planificación a posteriori, se ha intentado reflejar la idea que se tenía al iniciar el proyecto. Durante el proyecto, hubo desviaciones por distintos motivos, que a continuación se comentan por encima.

En primer lugar, los compromisos laborales externos al proyecto supusieron que algunos de los días que están en la planificación no se dedicaron al proyecto. Por otro lado, la dedicación a las tareas formativas fue incluso mayor de lo planificado: la biología en ese entonces era un campo muy nuevo para mí, y quería comprender bien lo que estaba haciendo. Se estima que en promedio se dedicó un 20% más de lo previsto a tareas de formación, lo cual equivale a un total de 52 horas.

En cuanto a la implementación del ranking (tarea I2), se consideró que la implementación inicial era mejorable y se hizo otra implementación, que fue la definitiva. Eso supone la creación de una nueva tarea (I2.2) con una dedicación de 25 horas. Además, se implementaron un número considerable de funcionalidades y opciones de salida adicionales (tarea IC1), cosa que supuso una dedicación de 20 horas más.

Todas estas desviaciones hicieron que la parte de implementación del proyecto se alargase hasta junio de 2014, Se tomó entonces la decisión de no implementar la integración con un sistema de generación de gráficos (tarea IC2), ya que esto se consideraba una funcionalidad adicional y hubiese requerido alargar el proyecto aún más.

Finalmente, mencionar que la mayor parte de la fase de documentación (tarea GD) se ha realizado entre agosto y diciembre de 2019.

No hubo cambios en la metodología propuesta, se consideró que era la que mejor se adaptaba al planteamiento del proyecto.

6. PRESUPUESTO

En cuanto al presupuesto de este proyecto, hay que tener en cuenta que es una aproximación, no se conocen los modelos exactos del equipo que se usó en el PRBB, y tampoco los factores de economía de escala que reducen los costes en un centro de investigación del tamaño del PRBB. Además, hay que tener en cuenta que no se cobró el salario por mano de obra, debido al carácter de la relación con el IMIM, de colaborador externo no remunerado, por lo tanto el coste de mano de obra no se tradujo en un gasto monetario para el IMIM, así que el único gasto para el IMIM fueron los costes de funcionamiento y mantenimiento del hardware que se usó.

6.1 DISCUSIÓN Y CÁLCULO DE LOS ELEMENTOS DEL PRESUPUESTO

El primer elemento a incluir en el presupuesto es el coste de la mano de obra. Cabe preguntarse si las horas de implementación deberían tener un coste por hora distinto a las de formación o documentación. Sin embargo, debido a que la formación fue mayoritariamente autónoma y no conllevó una dedicación importante de recursos del PRBB, se considera que la dedicación a una formación necesaria para el desarrollo del proyecto debería tener el mismo coste que la mano de obra de las tareas de implementación. Por esta razón y para simplificar, se aplica un coste estándar de 8€/h por mano de obra para la dedicación global estimada del proyecto, Así, el cálculo del coste por mano de obra es el siguiente;

Mano de obra: 550h · 8€/h = 4400€

Como parte del control de riesgos, podemos aplicar una partida de contingencia del 20% sobre el coste de la mano de obra para cubrir todas las desviaciones en la dedicación estimada en la implementación.

Partida de contingencia: 4400€ · 0,2 = 880€

La inmensa mayoría de las tareas, incluidas las de formación y documentación, requieren el uso de un terminal PC. Así, en primer lugar, consideraremos el coste de uso y desgaste del PC. Consideraremos que el PC se amortigua en 5 años, y estimamos un valor de compra de 250€ de gama media de oficina para un PC con todos los periféricos (aunque posiblemente, por cuestiones de escala, el PRBB podría conseguir un precio considerablemente menor). Además, consideramos un uso de 1 año para este proyecto.

Coste de uso y desgaste del terminal PC: 250€/5 = 50€

A continuación, computaremos el coste del gasto eléctrico del uso del PC, considerando un consumo de unos 220W de un PC de gama media con un monitor.

Gasto eléctrico total del uso del PC: 550h * 220W = 121kWh

Aplicamos un coste de kWh de 0.13€/kWh

Coste eléctrico del uso del terminal PC: 121kWh · 0,13,€/kWh = 15,73€

Coste total del uso del terminal PC: 15,73€ + 50€ = 65,73€

Además del uso del terminal PC, hay que tener en cuenta el coste del uso servidor. La mayoría de tests se harán en el terminal PC y solo para las funcionalidades más importantes computacionalmente se hará algún test con los datos completos en el servidor, además de los tests finales, que también se harán en el servidor. Teniendo esto en cuenta, hacemos una estimación con bastante margen de 40 horas de uso de servidor. A partir de esa estimación, se hacen dos cálculos del coste. La primera estimación del coste, probablemente sea una cota superior, será tener en cuenta el coste de mantenimiento: electricidad y el contrato del técnico de mantenimiento. Consideramos que el coste original del servidor de un nodo del servidor en unos 2000€, y un periodo de amortiguación de 8 años.

Así, el coste de uso y desgaste de ese nodo del servidor correspondiente a este proyecto es:

$$40h \cdot \frac{2000€}{8años} \cdot \frac{1año}{8766h} = 1{,}14€$$

Para calcular la parte de coste correspondiente al contrato del técnico de mantenimiento, asumiremos un contrato de 35000€ al año. En un centro de investigación de la escala del PRBB, podemos considerar que un técnico puede estar al cargo del mantenimiento de varios servidores, y haremos la suposición que entre todos los servidores asignados al técnico hay una media de 10 usuarios ejecutando concurrentemente en cada momento. Con estas premisas tenemos:

Coste asociado al contrato del técnico de mantenimiento del servidor:

$$\frac{40h \cdot \frac{35000€}{1año} \cdot \frac{1año}{8766h}}{10} = 15{,}97€$$

El gasto energético de un único servidor científico podemos suponer que es de un 200% de un PC normal, pero podemos asumir que de media hay 3 usuarios ejecutando concurrentemente en un único servidor. De esta manera tenemos que el coste eléctrico del uso del servidor será:

Coste eléctrico asociado al uso del servidor:

$$\frac{40h \cdot 0{,}44kW \cdot 0.{,}13€/kWh}{3} = 0{,}76€$$

Coste total del uso del servidor: 15.97€ + 0,76€ + 1,14€ = 17,87€

Por otro lado, se ha hecho una segunda estimación en el precio de 40 horas de computación en Microsoft Azure en una única máquina virtual con 16 núcleos y 64GB de RAM (instancia D16as v3), considerando que seguramente es comparable, o probablemente más potente, que el servidor del PRBB.

Coste de 40 horas de computación en Microsoft Azure: 14,98€

Consideramos entonces que nuestra primera estimación es razonable y la damos por buena.

La herramienta principal usada en el desarrollo es NetBeans, que es de descarga gratuita, de manera que no hay costes asociados a herramientas de desarrollo u otro software.

6.2 PRESUPUESTO FINAL

Concepto	Importe
Mano de obra	4400€
Partida de contingencia	880€
Uso del terminal PC	65,73€
Uso del servidor	17,87€
Total	**5363,60€**

Tabla 2. Presupuesto. *Elaboración propia*

7. ANÁLISIS DE SOSTENIBILIDAD

A continuación, se hace un pequeño análisis de sostenibilidad en distintos campos.

- **Sostenibilidad ambiental**: En un proyecto de desarrollo de software como este, no se generan residuos físicos. La única consideración ambiental que se puede hacer es el gasto energético, relativamente pequeño, que se hace a la hora de ejecutar el programa. En este sentido, se podría considerar que cuanto más eficiente sea la programación, serán necesarios menos recursos computacionales para obtener el mismo resultado, y, por lo tanto, menor será el impacto ambiental. Así, se podría decir que el rendimiento computacional, ya de por si un factor importante en la implementación, esta correlacionado con la sostenibilidad ambiental.
- **Sostenibilidad económica**: este proyecto se enmarca en un proyecto científico, que en principio no tiene como objetivo generar un beneficio económico directo. En este tipo de proyectos, la idea es usar los recursos asignados de la mejor manera para conseguir avances en el ámbito científico. En este sentido, el hecho de que el coste del proyecto no se traduzca en un gasto real para el grupo de investigación hace que los objetivos de sostenibilidad económica se cumplan sobradamente. Y aun teniendo en cuenta el coste estimado en el presupuesto que se presenta en este documento, no parece un coste excesivo para un desarrollo e implementación de un software de estas características.
- **Sostenibilidad social**: se deben hacer dos consideraciones en este sentido. En primer lugar, el desarrollo de este software responde a una investigación científica sobre genética del cáncer y, por lo tanto, cualquier avance que se haga en la materia tiene impacto social, aunque sea indirecto. Por otra parte, el software desarrollado en un proyecto científico público se suele publicar para ponerlo a disposición de otros proyectos científicos que podrían beneficiarse directamente o usar el código como base para adaptar el software a sus necesidades particulares.
- **Sostenibilidad técnica**: quizá este sea el aspecto de sostenibilidad más relevante en un proyecto software de estas características. Como ya se ha comentado, la decisión de hacer la implementación en Java corresponde en parte a este criterio: Java se considera algo más accesible a programadores medios que C++, y por lo tanto, si en algún equipo científico podrían beneficiarse de hacer modificaciones al código, es más probable que lo puedan hacer en Java. Aun así, priorizar la robustez es importante en un ambiente en el que muchos científicos tienen conocimientos de programación limitados, y por lo tanto la corrección de errores puede no ser fácil.

8. DISEÑO DE CLASES Y ESPECIFICACIÓN

La decisión de implementar el sistema en Java obliga a usar el paradigma de orientación a objetos, ya que en Java todo se programa en base a clases. Las características de este proyecto, la implementación de un algoritmo que trabaja sobre una estructura de datos grande pero sencilla (una matriz con valores reales que luego se transforma a una matriz de rankings enteros), no requeriría una estructura de clases. Sin embargo, dado que es un requisito de Java, se propone una estructura de clases sencilla que ayude a compartimentar el código, en función de los datos que se necesiten en cada momento. Además, se aplica una estructura por capas no estricta, con clases especializadas para leer y escribir los ficheros.

8.1 IDEA GENERAL DE LA IMPLEMENTACIÓN DEL ALGORITMO

Antes de describir las clases, se explicará la idea general de la implementación que se hizo del algoritmo kTSP.

- En primer lugar, se lee el fichero de datos de entrada: se guardan por separado las cabeceras de las filas (identificadores de genes o isoformas), y la matriz de datos, mientras que la cabecera de columnas (identificador de la muestra) no se guarda, solo se usa para segregar las columnas en muestras normales o tumorales (u otras dos clases o condiciones que interese diferenciar).
- Se crea la matriz de ranking a partir de los datos originales. Para cada columna, se crea una lista de tuplas con el valor y la posición original y se ordena la columna, mapeando luego el orden en la posición original en la nueva matriz de ranking.
- Se busca la mejor k (número de pares de genes o isoformas que se usan en el modelo predictivo). Para ello, se particionan las muestras (tanto normales como tumorales) y para cada partición independiente se hace una iteración: se calculan los *scores* de todas las parejas posibles y se guardan en un árbol ordenado (para cada gen, solo se usa la pareja con mayor *score*). A continuación, para cada k impar entre 1 y kmax se calcula el rendimiento de la predicción sobre la misma partición. Finalmente se elige la mejor k, calculando el rendimiento medio en todas las iteraciones para cada k.
- Finalmente, se selecciona el modelo con la k fijada calculando otra vez los *score* para cada pareja de genes o isoformas, esta vez sobre todos los datos, y eligiendo las k parejas con el mejor score (sin repetir genes o isoformas).
- Para cada pareja de genes o isoformas se calculan dos valores: el score y el rscore. El score se calcula sumando el porcentaje de muestras tumorales en el que el ranking de expresión del primer gen o isoforma es mayor que el segundo y el porcentaje de muestras normales en que el segundo gen es mayor en ranking. Es decir, mide si hay un intercambio de órdenes de expresión entre los dos genes que sea consistente entre la condición tumoral y la normal (entre 0 y

2, con el valor de 1 si no hay ningún intercambio de ordenes). El rscore es una medida secundaria, y mide la diferencia promedio del promedio de las diferencias de los rankings de la pareja de gen para las muestras. Es decir, es una medida de la magnitud del intercambio de órdenes. El score es la medida principal y el rscore es una medida secundaria que se usa para desempatar si dos parejas tienen el mismo score. En la versión para isoformas, solo se comparan parejas de isoformas que sean del mismo gen.

- El rendimiento de predicción se calcula en base al modelo dado. Para cada muestra, se miran las parejas de genes o isoformas del modelo, si el primer gen es mayor al segundo, esa pareja asigna esa muestra a tumoral, y si no, a normal. La asignación definitiva es aquella que concuerde con más parejas (por eso no se admiten valores de k pares… en la predicción no puede haber empates). Se mira el número de predicciones correctas e incorrectas tanto en muestras normales como en tumorales (para decidir la mejor k, se usa como rendimiento de la predicción el porcentaje de aciertos en la predicción).
- Además del algoritmo descrito arriba, el programa incluye distintas opciones. Las más destacables son la opción de hacer la predicción y calcular el rendimiento de predicción para un modelo dado, o hacer la selección del modelo con la división en clases de la muestra redistribuida aleatoriamente (para poder tener un control negativo).

8.2 ESPECIFICACIÓN DE LA ENTRADA, DE LA SALIDA Y OPCIONES

El programa, en cualquiera de los modos de ejecución, requiere un argumento obligatorio, con el nombre o el *path* del fichero con los datos de la transcripción de los genes o isoformas. El formato de este fichero debe ser una tabla o matriz con datos numéricos decimales y separados por tabuladores. La primera fila debe tener los identificadores de las muestras, que deben tener como sufijo el identificador de la clase o condición a diferenciar (“N” y “T” por defecto para muestras normales y tumorales). Para cada línea de datos, la primera columna debe contener el identificador del gen o, en el caso del modo para isoformas, el identificador de la isoforma, que contenga el identificador del gen de esa isoforma en el formato “id_gen,id_isoforma”.

El fichero de salida es un fichero de texto plano. Cada línea del fichero contiene información sobre algún parámetro de la ejecución o una de las parejas del modelo final que da el software en sus modos principales de ejecución, La información que se incluye en el fichero depende del modo de ejecución y de las opciones especificadas. Al principio de cada línea aparece la iteración a la que hace referencia la línea y el tipo de información que da la línea. Las líneas más relevantes son aquellas que empiezan con “final model_pair”, que son las que especifican los pares de genes o isoformas que forman el modelo predictivo que ha elaborado el programa.

Además del fichero de datos, se pueden incluir opciones como argumentos. A continuación, se describen todas las opciones que se pueden incluir como argumentos.

- -h: imprime la ayuda del programa.
- -i: el programa se ejecutará en el modo para isoformas.
- -o: (seguido de un nombre de fichero) especifica el nombre del fichero de salida. Si no se especifica, el fichero de salida tendrá el mismo nombre que el de entrada, con un sufijo indicando el modo en el que se ha ejecutado el programa.
- -n: (seguido de un entero positivo): especifica el número de iteraciones (y indirectamente el tamaño de la partición) en la primera fase del algoritmo, en que se busca la mejor k. Por defecto es 10.
- -k: (seguido de un entero positivo): define el valor máximo de la k, el parámetro kmax del algoritmo. Se admiten valores pares, aunque se usará siempre el mayor valor impar dentro del rango admitido (tanto al usar –k 10 como –k 9 resulta en que el programa usará como kmax el valor 9). Por defecto es 10.
- -s: (seguido de un entero positivo) define el número de los mejores pares de genes o isoformas del que se escribirá los scores. Esto permite obtener información de los mejores pares más allá de los seleccionados para el modelo final. No tiene efecto si el número es menor que la k óptima seleccionada por el algoritmo, porque la información sobre los pares del modelo final siempre se escribe.
- -c: (seguido de dos *strings* separados por espacio) define los sufijos que se emplean en los nombres de las muestras para diferenciar entre las clases o condiciones. Por defecto son “N” y “T”.
- -p: (seguido por el nombre o el *path*) del fichero que contiene el modelo de predicción) ejecuta el programa en el modo de solo predicción. Permite ver como de bien predice los datos de entrada un modelo creado por el usuario o sacado de una ejecución anterior. El formato del fichero que contiene el modelo debe contener en cada línea un par de identificadores de genes o isoformas separados por espacios. El número de pares de genes o isoformas en el fichero debe ser impar.
- -d: se imprimirán los detalles de la predicción para cada muestra, con el número de pares que predicen correctamente e incorrectamente esa muestra. Se usa solo con -p y -r.
- -l: (seguido de un entero positivo): se hará el algoritmo con una k = 1 con la distribución de las muestras en clases o condiciones hecha de manera aleatoria. Se repetirá el número de veces especificado. Esta opción se usa como control negativo.
- -r: (seguido de un entero positivo): se ejecutará el modo de solo predicción, seleccionando un modelo al azar de entre los genes o isoformas presentes en el fichero de datos. El modelo tendrá el número de pares de genes o isoformas especificado (debe ser impar). Esta opción se usa como control negativo.

- --seed: (seguido de un entero): permite especificar la semilla para el algoritmo de generación de números aleatorios que usa Java en su librería. Si no se específica, se usará la función de la librería de Java para seleccionar la semilla.

En algunos de los modos de ejecución, algunas de las opciones no tienen sentido. Por lo tanto, hay una preferencia hacia las opciones que definen el modo de ejecución, y en caso de que se incluya alguna opción que no tenga sentido, se ignora y se escribe un *warning* en el canal de error.

8.3 DESCRIPCIÓN DE LAS CLASES Y FUNCIONES

A continuación, se hace una primera especificación de las clases y las funciones, con una breve descripción de su propósito. Para las funciones, usaremos la forma de pre y post, para hacer una especificación más formal.

Clase **Main**: es la clase principal, que contiene la función main, que se ejecuta al iniciar el programa y la clase printHelp que escribe por pantalla las instrucciones de uso del programa.

- Función **main**(args): esta función se ejecuta al iniciar el programa, procesa los argumentos (dando errores y *warnings* cuando corresponde) y se encarga de construir la matriz de ranking a partir de los datos iniciales antes de llamar a la función de la clase Ktsp correspondiente para hacer los pasos principales del algoritmo.
    - Pre: se ha iniciado la ejecución del programa con los argumentos args.
    - Post: se ha ejecutado el programa con las opciones indicadas por args y se ha escrito el resultado en el fichero correspondiente. En caso de error, se ha escrito por pantalla un mensaje especificando el motivo del error.
- Función **printHelp**():esta función se ejecuta si se ha usado la opción –h como argumento. Escribe por pantalla una ayuda describiendo el uso y las opciones del programa.
    - Pre: se ha usado la opción –h en los argumentos al ejecutar el programa
    - Post: se ha escrito por pantalla el texto de ayuda.

Clase **Ktsp**: esta clase contiene los datos para ejecutar los pasos principales del algoritmo y las funciones que ejecutan el algoritmo en los distintos modos, así como algunas funciones auxiliares, internas de la clase.

- Atributo **ranks**: matriz de ranking que contiene los rankings de expresión de cada gen o isoforma en cada muestra.
- Atributo **normal**: lista de los índices (dentro de la matriz ranks) de las muestras de tejido normal (o de la primera clase o condición a comparar)
- Atributo **tumoral**: lista de los índices de las muestras de tejido normal (o de la segunda clase o condición a comparar)

- Atributo **idGene**: lista con los identificadores de los genes o isoformas que hay en la matriz ranks,
- Atributo **writer**: instancia de la clase FileWriter, para poder escribir los resultados en el fichero de salida.
- Función **Ktsp**(r, n, t, g, w): constructor que inicializa los atributos con los valores que se pasan como argumentos.
    - Pre: r, n, t, g, w son instancias validas de los atributos de la clase correspondientes
    - Post: se ha creado una instancia de la clase Ktsp, inicializando los atributos con los valores de los argumentos de la función.
- Función **runKtspWithCrossvalidation**(kmax, iterations, show, isoforms): ejecuta el modo estándar del algoritmo ktsp, ya sea en su versión original para genes o en la versión específica para isoformas. Este modo busca la mejor k iterando sobre particiones de las muestras, tanto tumorales como normales, y una vez encuentra la mejor k, hace un último paso del algoritmo con la mejor k sobre todos los datos y obtiene el modelo, escribiendo, si corresponde, los resultados parciales en un fichero.
    - Pre: kmax es el k máximo, o número de parejas de genes (o isoformas) que tendrá el modelo final resultante del algoritmo, iterations es el número de iteraciones que se hacen para encontrar el mejor óptimo de k, show es el número de pares que se quieren escribir en el fichero con su puntuación (posiblemente mayor que k), isoforms indica si se ejecuta la versión del algoritmo para isoformas o para genes.
    - Post: se ha ejecutado el algoritmo ktsp completo, se han escrito en el fichero de salida los resultados (pares de genes o isoformas y su puntuación), y se retorna el modelo de predicción resultante.
- Función **runKtspWithRandomisedLabels**(iterations, isoform, random): ejecuta varias veces el paso de aprendizaje del algoritmo (sin los pasos para determinar la mejor k) con k = 1, y con las muestras redistribuidas aleatoriamente entre las clases (normal y tumoral por defecto). Esto se usa como un control negativo.
    - Pre: iterations es el número de iteraciones que se quiere repetir el test aleatorio, isoform indica si se usará la versión del algoritmo para isoformas o para genes, y random es una instancia de la clase Random de Java.
    - Post: se ha escrito en el fichero de salida, para cada iteración, la mejor pareja y su puntuación haciendo el análisis con la distribución de muestras aleatoria.
- Función **runSingleKtsp**(kmax, tStartNormal, tendNormal, tStartTumor, tEndTumor): ejecuta una iteración del algoritmo ktsp para genes, sobre la partición de las muestras indicada, haciendo todas las comparaciones para las

posibles parejas de genes. A continuación, se busca la mejor k mientras se hace el testing sobre la misma partición (con un porcentaje de predicción).

  - Pre: kmax es la k máxima, y tStartNormal, tEndNormal, tStartTumor, tEndTumor son las posiciones en las listas normal y tumor que delimitan la partición.
  - Post: se han escrito en el fichero de datos los resultados de la iteración y se retorna el modelo resultante de esa iteración.

- Función **runSingleKtspIso**(kmax, tStartNormal, tendNormal, tStartTumor, tEndTumor): ejecuta una iteración del algoritmo ktsp para isoformas. La única diferencia es que no se comparan todas las parejas posibles de isoformas, sino solo se hacen comparaciones entre isoformas del mismo gen.
  - Pre: kmax es la k máxima, y tStartNormal, tEndNormal, tStartTumor, tEndTumor son las posiciones en las listas normal y tumor que delimitan la partición.
  - Post: se han escrito en el fichero de datos los resultados de la iteración y se retorna el modelo resultante de esa iteración.
- Función **fixedKSelection**(k, show): ejecuta el último paso del algoritmo ktsp para genes. La única diferencia es que no se comparan todas las parejas posibles de isoformas, sino solo se hacen comparaciones entre isoformas del mismo gen.
  - Pre: k es la mejor k, seleccionada en pasos anteriores, y show es el número de parejas a escribir en el fichero de salida (puede ser mayor que k).
  - Post: se devuelve el modelo final seleccionado (con pares adicionales si show es mayor que k).
- Función **fixedKSelectionISO**(k, show): ejecuta el último paso del algoritmo ktsp para isoformas. Una vez seleccionada la mejor k, se hace la selección del modelo haciendo las comparaciones sobre todas las muestras, sin particiones.
  - Pre: k es la mejor k, seleccionada en pasos anteriores, y show es el número de parejas a escribir en el fichero de salida (puede ser mayor que k).
  - Post: se devuelve el modelo final seleccionado (con pares adicionales si show es mayor que k).
- Fuunción **testWithSinglePair**(pair): ejecuta la predicción sobre todos los datos con un solo par de genes o isoformas, mirando el porcentaje de aciertos en la predicción y calculando otros parámetros, como el *information gain*.
  - Pre: pair es una pareja de genes o isoformas que está presente en la matriz de datos.
  - Post: se escribe en el fichero de salida el resultado del testeo con la pareja dada, e información adicional.

- Función **testModel**(model, printDetails): ejecuta la predicción sobre todos los datos con un modelo ktsp ya dado. Se basa en el porcentaje de aciertos en la predicción del modelo dado.
    - Pre: model es un modelo válido, formado por pares de genes o isoformas que están presentes en la matriz de datos, y printDetails indica si se deben escribir todos los detalles del proceso de predicción
    - Post: se ha escrito en el fichero de salida los resultados de la predicción, basados en el porcentaje de aciertos, y si se corresponde, también se ha escrito los detalles del proceso de predicción.

Clase **DataWithIndex**: esta clase funciona simplemente como una tupla, conteniendo un valor de la matriz de datos original y su posición original, antes de ordenar la columna. Se usa para construir la matriz de rankings.

- Atributo **data**: valor en la matriz de datos original
- Atributo **index**: posición dentro de la columna de la matriz original
- Función **DataWitzIndex**(d, i): constructora de la clase que inicializa los atributos con los valores de los parámetros de la función.
    - Pre: d, i son instancias validas de los atributos de la clase correspondientes
    - Post: se ha creado una instancia de la clase DataWithIndex inicializando los atributos con los valores de los argumentos de la función.
- Función **compareTo**(other): implementa la función compareTo de la *interface* Comparable, para que la comparación se haga a partir del atributo data. Implementar esta función es un requisito para que la clase sea ordenable.
    - Pre: other es la instancia de DataWithIndex con la que se quiere comparar.
    - Post: se devuelve el valor de la comparación entre los atributos data.

Clase **PairRankInfo**: esta clase funciona simplemente como tupla, conteniendo la información de una pareja de genes o isoformas y su score en el algoritmo ktsp.

- Atributo **firstIndex**: la posición dentro de la matriz de ranking del primer gen o isoforma de la pareja.
- Atributo **secondIndex**: la posición dentro de la matriz de ranking del segundo gen o isoforma de la pareja.
- Atributo **score**: el score de la pareja de genes o isoformas, refleja si hay intercambios consistentes en los rankings de los genes entre las condiciones tumoral y normal.
- Atributo r**score**: el rscore de la pareja de genes o isoformas, una medida secundaria que refleja la magnitud de la diferencia promedio entre los genes o isoformas de la pareja.

- Función **PairRankInfo**(fi, si, s, rs): constructora de la clase que inicializa los atributos con los valores de los parámetros de la función.
    - Pre: fi, si, s i rs son instancias validas de los atributos de la clase correspondientes
    - Post: se ha creado una instancia de la clase PairRankInfo inicializando los atributos con los valores de los argumentos de la función.
- Función **compareTo**(other): implementa la función compareTo de la *interface* Comparable, para que la comparación se haga a partir del atributo score, y en caso de empate, a partir del atributo rscore. Implementar esta función es un requisito para que la clase sea ordenable.
    - Pre: other es la instancia de pairRankInfo con la que se quiere comparar.
    - Post: se devuelve el valor de la comparación entre los atributos score, y si hay empate, entre los atributos rscore.

Clase **KtspPredictionModel**: esta clase funciona simplemente como un contenedor que tiene la información de un modelo predictivo: los pares de genes o isoformas ordenados por su score y los resultados del rendimiento de la predicción (para cada k) una vez realizada.

- Atributo **pairs**: lista de los pares de genes o isoformas que tiene el modelo.
- Atributo **testingPerformance**: lista de los rendimientos de las k (solo las k impares) cuando se hace el paso de selección de la mejor k.
- Función **KtspPredictionModel**(): constructora de la clase que inicializa los atributos a listas vacías.
    - Pre: -
    - Post: se ha creado una instancia de la clase KtspPredictionModel inicializando sus atributos como listas vacías.

Clase **FileReader**: Clase encargada de leer el fichero de entrada que tiene los datos de expresión de los genes o isoformas. Esta clase lee las líneas de datos por petición y no guarda las líneas anteriores.

- Atributo **currentDataLine**: lista con los datos de la última línea que se ha leído del fichero de entrada.
- Atributo **sampleIdList**: lista con los identificadores de las muestras, corresponde a la primera línea del fichero de entrada.
- Atributo **geneIdList**: lista con los identificadores de los genes o isoformas leídos hasta ese momento.
- Atributo **countlines**: número de líneas leídas hasta ahora del fichero de entrada.
- Atributos **br**, **fis**, **dis**: instancias de las clases de java BufferedReader. FileInputStream y DataInputStream, que se usan para leer el fichero de entrada.

- Función **FileReader**(filename): constructora de la clase que abre el fichero de entrada y lee la primera línea (con los ids de las muestras), parseandola y guardándola.
    - Pre: filename es el nombre del fichero de entrada.
    - Post: se ha creado una instancia de la clase FileReader, se han inicializado los atributos, se ha abierto el fichero de entrada con nombre indicado por el argumento. Se ha leído la primera línea, se ha incrementado countlines y se ha guardado los ids de las muestras en el atributo sampleIdList.
- Función **getSampleNames**(): función para obtener los ids de las muestras.
    - Pre: se ha abierto el fichero de entrada y se ha leído correctamente la primera línea.
    - Post: se retorna la lista de ids de las muestras.
- Función **getNextLine**(): función para leer la siguiente línea de datos del fichero de entrada-
    - Pre: se ha abierto el fichero de entrada y se ha leído correctamente al menos la primera línea.
    - Post: se lee la siguiente línea del fichero de entrada, se incrementa countlines, se parsea, se añade la primera columna (correspondiente al id del gen o isoformas (a geneIdList y se retorna la lista con el resto de datos de la línea.
- Función **getGeneNames**(): función para obtener la lista de identificadores de genes o isoformas de las líneas leídas hasta el momento. Está pensada para usarse una vez leídas todas las líneas, para obtener la lista de completa de genes o isoformas.
    - Pre: se ha abierto el fichero de entrada y se ha leído correctamente al menos la primera línea.
    - Post: se retorna la lista con los identificadores de genes o isoformas de las líneas leídas hasta el momento.

Clase **ModelFileReader**: Clase encargada de leer el fichero de entrada que contiene un modelo para usar en el modo de predicción.

- Atributo **model**: contiene el modelo de predicción ktsp que se ha leído del fichero de entrada.
- Atributo **idGene**: lista con los identificadores de los genes o isoformas leídos en el fichero de datos (no es el mismo que el del modelo, que lee esta clase).
- Atributos **br**, **fis**, **dis**: instancias de las clases de java BufferedReader. FileInputStream y DataInputStream, que se usan para leer el fichero de entrada.
- Función **FileReader**(filename, genes): constructora de la clase que abre el fichero de entrada y lo lee, parseandolo y guardando el modelo de predicción resultante. Comprueba que los genes o isoformas del modelo estén presentes en el fichero de datos.

    - Pre: filename es el nombre del fichero de entrada con un modelo de predicción y genes es la lista de identificadores de genes o isoformas que se ha leído del fichero de datos principal.
    - Post: se ha creado una instancia de la clase ModelFileReader, se ha inicializado idGene con el argumento correspondiente de la función, se ha abierto el fichero de entrada con nombre indicado por el argumento. Se ha leído y parseado el fichero, creando y guardando el modelo de predicción resultante.
- Función **getModel**(): función para obtener el modelo de predicción leído del fichero.
    - Pre: se ha abierto el fichero de entrada y se ha leído correctamente el modelo.
    - Post: se retorna el modelo de predicción leído.
    - das hasta el momento.

Clase **FileWriter**(): Clase encargada de escribir los distintos resultados en el fichero de salida. Tiene funciones para escribir todos los tipos de información en el formato de salida correspondiente. Hay que tener en cuenta que la<a escritura en el fichero se hace progresivamente, y en distintas partes del código, a medida que se van obteniendo los resultados (no se difiere la escritura al final del programa). Por lo tanto, si el programa no acaba correctamente, puede haber resultados parciales escritos en el fichero de salida.

- Atributo **iteration**: contador de iteraciones en el proceso de buscar la mejor k particionando las muestras.
- Atributo **outputFile**: instancia de la clase PrintWriter de Java encargada de escribir en el fichero.
- Atributo **class1**: nombre de la etiqueta de la primera clase o condición que se pretende distinguir (coincide con el sufijo distintivo de la clase en los identificadores de las muestras).
- Atributo **class2**: nombre de la etiqueta de la segunda clase o condición que se pretende distinguir (coincide con el sufijo distintivo de la clase en los identificadores de las muestras).
- Atributo **sampleNames**: lista de los identificadores de las muestras, tal y como se han leído en el fichero de datos.
- Atributo **df**: instancia de la clase DecimalFormat de Java, para especificar el formato de escritura de los decimales, y en concreto el número de decimales que se escriben después de la coma.
- Función **FileWriter**(filaneme, c1, c2, sn): constructora de la clase que inicializa todos los atributos y crea el fichero de salida.
    - Pre: filename es el nombre del fichero de salida, c1 y c2 son los identificadores de las clases a diferenciar (que aparecen como sufijos en

los identificadores de las muestras) y sn es la lista de identificadores de las muestras leído del fichero de entrada.
    - Post: se ha creado una instancia de la clase FileWriter, inicializando los atributos con los argumentos correspondientes y se ha abierto un fichero de escritura con el nombre indicado.
- Función **closeFile**(): esta función cierra el fichero de salida una vez se ha acabado de escribir todos los resultados.
    - Pre: se ha abierto un fichero de salida.
    - Post: se ha cerrado el fichero de salida asociado a la instancia de la clase.
- Función **nextIteration**(): incrementa el contador de iteraciones.
    - Pre; -
    - Post: se ha incrementado el contador de iteraciones.
- Función: **printIterationKmaxPair**(first, second, score, rscore): escribe en el fichero de salida una línea con la información de una pareja asociada a una iteración. Se usa para escribir las kmax mejores parejas seleccionadas en cada iteración.
    - Pre: el fichero de salida está abierto para escritura. first, second, score y rscore son la información de una de las kmax mejores parejas de la iteración, y el atributo iteration tiene el valor correspondiente a la iteración actual.
    - Post: se ha escrito en el fichero de salida la información de la pareja, asociándola con la iteración actual.
- Función **printIterationKPerformance**(k, TN, TT, FN, FT): escribe en el fichero de salida una línea con la información del rendimiento de la predicción para una k en una iteración concreta.
    - Pre: el fichero de salida está abierto para escritura. k es el valor de la k asociado a los resultados. TN, TT, FN, FT corresponden al número de predicciones acertadas y falladas para las clases o condiciones a diferenciar para una k y una iteración, y el atributo iteration tiene el valor correspondiente a la iteración actual.
    - Post: se ha escrito en el fichero de salida la información del rendimiento de la predicción, para una k en la iteración actual.
- Función **printKAveragePerformance**(k, performance): escribe en el fichero de salida una línea con la información del rendimiento de predicción medio para una k concreta en todas las iteraciones.
    - Pre: el fichero de salida está abierto para escritura. k es el valor de la k asociado a los resultados. performance corresponden al promedio del porcentaje de aciertos para una k entre todas las iteraciones.
    - Post: se ha escrito en el fichero de salida la información del promedio del rendimiento de la predicción, para una k en todas las iteraciones.

- Función **printSinglePairPerformance**(first, second, TN, TT, FN, FT, IG, score, rscore): escribe en el fichero de salida una línea con la información del rendimiento de la predicción para un único par de genes o isoformas en el conjunto de todos los datos.
    - Pre: el fichero de salida está abierto para escritura. First y second son los genes o isoformas usados para la predicción, score y rscore son los valores para este par. TN, TT, FN, FT corresponden al número de predicciones acertadas y falladas para las clases o condiciones a diferenciar para este par y IG es el *information gain* calculado a partir de los resultados.
    - Post: se ha escrito en el fichero de salida la información del rendimiento de la predicción, para el par de genes o isoformas.
- Función **printFinalModelPair**(first, second): escribe en el fichero de salida una línea con un par de genes o isoformas pertenecientes al modelo de predicción seleccionado. Se usa multiples veces para escribir todo el modelo final.
    - Pre: el fichero de salida está abierto para escritura. First y second son los genes o isoformas de un par perteneciente al modelo final.
    - Post: se ha escrito en el fichero de salida el par de genes o isoformas del modelo final.
- Función **printModelPerformance**(TN, TT, FN, FT): escribe en el fichero de salida una línea con la información del rendimiento de la predicción para el modelo que se ha proporcionado en el modo de solo predicción o seleccionado en el paso final del algoritmo.
    - Pre: el fichero de salida está abierto para escritura. TN, TT, FN, FT corresponden al número de predicciones acertadas y falladas para las clases o condiciones a diferenciar para el modelo proporcionado en el modo de solo predicción.
    - Post: se ha escrito en el fichero de salida la información del rendimiento de la predicción, usando el modelo dado.
- Función **printIPredictionDetails**(sample, correctVotes, incorrectVotes): escribe en el fichero de salida una línea con la información de los detalles de la predicción para una muestra concreta para el modelo que se ha proporcionado en el modo de solo predicción o seleccionado en el paso final del algoritmo.
    - Pre: el fichero de salida está abierto para escritura. sample es la muestra de la que se está dando información y correctVotews e IncorrectVotes son el número de parejas del modelo que predicen de manera correctamente e incorrectamente esa muestra.
    - Post: se ha escrito en el fichero de salida los detalles de la predicción de una muestra concreta usando el modelo dado.
- Función **printRandomLabelResult**(score 1, score2): escribe en el fichero de salida una línea con la información del resultado en el score y el rscore de la mejor

pareja cuando se aplica el algoritmo con la distribución aleatoria de las muestras en las clases o condiciones a diferenciar.

- Pre: el fichero de salida está abierto para escritura. score1 y score2 son el score y el rscore respectivamente del mejor par seleccionado usando el modo aleatorio.
- Post: se ha escrito en el fichero de salida la información del mejor par seleccionado en el modo aleatorio.

8.4 DIAGRAMA UML

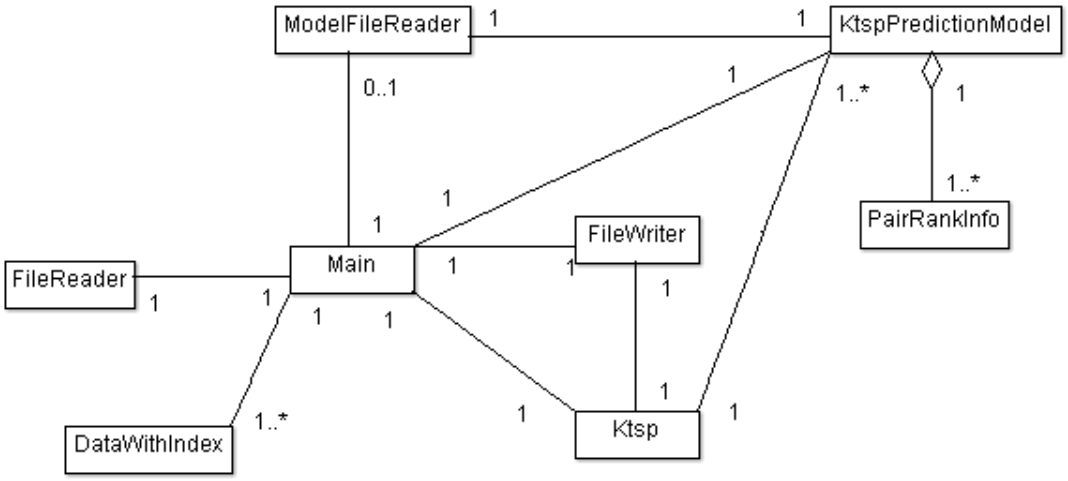

Fig. 3. Diagrama UML. *Elaboración propia*

Nota: en el diagrama, las asociaciones representan cualquier uso de una clase dentro de otra, incluida la pertenencia como atributo de la clase o la creación o uso de una instancia dentro de una función.

8.5 DETALLES RELEVANTES DE LA IMPLEMENTACIÓN

En este apartado no discutiremos paso a paso toda la implementación. Sin embargo, nos comentaremos algunos aspectos que consideramos más relevantes o que surgieron de decisiones importantes. Para consideraciones sobre el rendimiento temporal, definimos los siguientes símbolos:

- N el número de genes
- M el número de muestras
- I el número medio de isoformas de cada gen
- K es la k máxima
- J es el número de iteraciones que se hacen para encontrar la k óptima.

Hay que tener en cuenta que generalmente N >> M, N >> I, N >> K, M > J.

En primer lugar, para transformar los valores de la matriz original en la matriz de ranking, inicialmente se intentó plantear una solución distinta a la adoptada finalmente. Se intentó crear una matriz de listas encadenadas, una por cada columna, donde se hacía una ordenación por inserción, para luego recorrer la matriz creando la matriz de ranking definitiva. Con esta solución el coste computacional del primer paso es $O(M \cdot N^2)$. Al hacer los tests, no se consideró satisfactoria esta solución y se implementó la solución actual, que usa para cada fila, una lista de pares con el valor y la posición originales, ordenando la lista con la función *sort* de Java por el valor (que suponemos que es uno de los algoritmos de ordenación con coste O(n log n)), y luego recorriendo la lista ordenada y usando la posición original para crear la matriz de ranking. Esta solución fue más sencilla de implementar que la original, y es más eficiente, teniendo un coste de $O(M \cdot N \log N)$.

En segundo lugar, en los pasos principales del algoritmo ktsp, se decidió usar un TreeSet para poder mantener los pares que se comparan ordenados según el score y poder hacer inserciones ordenadas en O(log n). Tener en cuenta que limitamos los pares que guardamos a $K \cdot N$, ya que sabemos que no necesitaremos más. La parte más costosa del algoritmo son las comparaciones entre todos los pares posibles, para encontrar la mejor k. El coste estimado para el modo que trabaja con genes de esta parte del algoritmo es $O(J \cdot N^2 \cdot (M/J + \log (K \cdot N)))$. Considerando que $K \cdot N >> M/J$, el coste final es $O(J \cdot N^2 \cdot \log (K \cdot N))$.

Quizá la decisión más significativa de la implementación esta en la adaptación del algoritmo ktsp para trabajar con isoformas. El objetivo de esta adaptación es poder hacer modelos predictivos que se centren en cambios en la regulación del *splicing* alternativo. Por lo tanto, un modelo que considerase pares de isoformas que viniesen de distintos genes no estaría dando información sobre cambios en el *splicing* alternativo. Eso hace que la versión para isoformas de nuestro algoritmo solo compare entre si las isoformas que son del mismo gen. Este cambio hace que el coste computacional de la versión para isoformas del algoritmo sea mucho menor que la versión original. En concreto, el coste computacional de la versión para isoformas es $O(J \cdot N \cdot I^2 \cdot \log(K \cdot N)$.

Otro aspecto a comentar es que la implementación usa un diseño de clases por capas relajado. Tiene clases especializadas para escribir y leer los ficheros en los ficheros, cosa que permite cambiar o adaptar fácilmente el formato de lectura o escritura si se considera necesario. Por otro lado, la interfaz entre las capas no esta centralizada en una sola clase, las clases que necesitan escribir resultados tienen acceso a la clase de escritura correspondiente, de manera que se pueden escribir los resultados fácilmente a medida que se van generando. Finalmente, cabe mencionar que se ha hecho un esfuerzo en detectar y capturar los errores y excepciones más predecibles y escribir un mensaje informativo por el canal de error.

9. RESULTADO FINAL Y USO DENTRO DEL PROYECTO DE INVESTIGACIÓN

El resultado final de este trabajo fue la implementación del software descrito en el apartado anterior. Después de un testeo exhaustivo, tanto con *datasets* reducidos como con algunos de los *datasets* completos que pretendían usarse en el proyecto de investigación, se estableció que el software funcionaba de manera satisfactoria y en junio de 2014 se dio por finalizado el desarrollo.

En enero de 2015, se publica el articulo [6] fruto del proyecto de investigación. En él se presenta el análisis hecho de manera mayoritaria con el software iso-ktsp, tanto para obtener modelos predictivos como para hacer la validación de los resultados con las distintas opciones que provee el software. Esto hace aparente que el software provee un conjunto de herramientas bastante completo para las necesidades de este proyecto científico. La única de las funcionalidades inicialmente propuestas como opcional que no se ha llegado a implementar por falta de tiempo es la integración con alguna herramienta de generación de gráficos.

En el proyecto de investigación, Sebestyén y Eyras analizaron los datos de expresión de isoformas de 9 tipos de tumores diferentes emparejados con datos de expresión de tejidos normales. Entre los distintos resultados descritos en el artículo, destacamos algunos. Los modelos predictivos con pares de isoformas producidos para cada tipo de tumor son capaces de clasificar datos nuevos de tumores con más de un 84% de acierto. Si de esos modelos se cogen solamente los genes que tienen fuertes evidencias de tener cambios funcionalmente importantes, el porcentaje de acierto es de más de un 90%. Comparado con SwitchSeq, un software que hace un análisis parecido, los modelos dados por iso-kTSP son bastante mejores en cuanto a significación estadística. Además, se han usado los genes que aparecen en los modelos para hacer un catálogo de 244 cambios significativos en la regulación del *splicing* alternativo. De entre los genes involucrados en los modelos dados por el software, hay genes con una relación firmemente establecida previamente con el cáncer, pero también hay nuevos genes que podrían ser nuevos marcadores del cáncer. De manera interesante, en muchos casos los cambios de expresión de las isoformas no parecen tener una relación fuerte con mutaciones de los propios genes, cosa que indica cambios más complejos que afectan a los cambios de la regulación del *splicing* alternativo en el cáncer. En definitiva, el análisis hecho da un conjunto de genes que tienen cambios consistentes en la expresión de isoformas y que sirven para clasificar con un alto nivel de acierto las muestras tumorales. Esto presenta una base para futuros estudios centrados en estos genes, para comprender mejor las funciones celulares que están afectadas por los cambios y eventualmente mejorar las terapias contra el cáncer.

10. 5 AÑOS DESPUÉS: IMPACTO DEL ARTICULO Y USO DEL SOFTWARE

Aprovechando el hecho de que desde el desarrollo y la publicación del software hasta la redacción del presente documento han pasado más de 5 años, es interesante hacer una valoración del impacto que ha tenido el trabajo realizado. El software se publicó en el repositorio de herramientas de análisis de ARN de la Universitat Pompeu Fabra (UPF), con una licencia de uso y modificación libres (disponible en https://bitbucket.org/regulatorygenomicsupf/iso-ktsp).

En octubre de 2016, la Biblioteca Rector Gabriel Ferreté, de la UPC, publicaba un estudio [7] en el que incluía el artículo asociado a este software entre los 25 artículos más citados con autores de la UPC de 2015. En ese estudio se indica que el articulo tenía 14 citaciones según el recuento de Web of Science, 15 según Scorpio y 21 según Google Scholar. En enero de 2020, el recuento de citaciones que da Web of Science [8] es de 68, según crossref [9] son 60 y según Google Scholar [10] son 108.

Entre las citaciones se encuentran múltiples casos de uso del software. Algunos ejemplos son un Trabajo de Final de Grado de Biología Humana de la UPF de Marina Reixachs i Solé [11] y el artículo de Guo et al. [12] donde se integra el software iso-kTSP en un pipeline más amplio.

En el marco de la recopilación de información sobre el uso del software para este documento se encontró también una consulta en el sitio web Biostars [13], donde un usuario del software encontró una excepción al probar el software con un *dataset* muy pequeño. Después de analizar el problema, se encontró que esto se debía a un error en la implementación de la comprobación del número de iteraciones respecto al número de muestras de la clase menos representada. Si el usuario intenta realizar un numero de iteraciones mayor que el número de muestras de la clase menos representada, es imposible hacer la partición y el programa acaba dando un mensaje informativo. El usuario intentaba ejecutar el programa con 3 muestras tumorales, al no indicar el número de iteraciones, el valor por defecto es 10, con lo cual debería haberse dado el mensaje de error correspondiente. Sin embargo, en el código, el valor por defecto se asigna después de la comprobación de este error, con lo que en caso de que no se indique explícitamente el número de iteraciones, la comprobación no detecta el error, y el programa falla por una excepción de java más adelante. Se ha informado al usuario de la razón del error para que pueda hacer una ejecución correcta de su *dataset* de prueba. Por el momento, no se ha corregido el error de implementación, porque no se considera muy urgente solucionarlo. Más adelante se valorará si contactar con la UPF para obtener acceso al repositorio y corregir la implementación de la comprobación de este error.

11. CONCLUSIONES Y VALORACIÓN PERSONAL

En el curso de este Trabajo de Fin de Grado se hizo un software a medida para cubrir las necesidades específicas de un proyecto de investigación en genética del cáncer. Para ello, partiendo de la descripción del algoritmo kTSP previamente publicada, se implementó en Java este algoritmo, así como una adaptación propia del algoritmo original para trabajar con isoformas que hemos llamado iso-kTSP. Además, se han añadido múltiples funcionalidades y modos de ejecución para cubrir necesidades de validación de los resultados. Todo ello supuso, por un lado, aplicar los conocimientos, técnicas y experiencia obtenida a lo largo de los 5 años de carrera en Ingeniería Informática, y, por otro lado, un esfuerzo considerable de formación complementaria en los ámbitos que no conocía: sobre todo biología y genética, pero también en *machine learning*.

Entre las distintas habilidades que se han puesto a prueba en este trabajo, cabe destacar la capacidad de comunicar y entender las necesidades del usuario y traducirlas a un diseño e implementación de software que las cubran, la capacidad de comprender un algoritmo por su descripción formal y de adaptarlo según las particularidades del problema que se pretende solucionar, la capacidad de establecer prioridades en la metodología y en las decisiones técnicas y hacer compromisos en función de esas prioridades y, finalmente, la determinación y el compromiso realizar el mejor trabajo posible a pesar de otros compromisos laborales que tenía en ese momento.

En cuanto a las lecciones aprendidas y los aspectos a mejorar, destacaría dos aspectos concretos. En primer lugar, a pesar del esfuerzo priorizando la robustez de la implementación del software, con testeos exhaustivos y pretendiendo dejar el proyecto en un estado que no requiriese un mantenimiento posterior, se ha encontrado un error de implementación en base a una consulta de un usuario 4 años después de la publicación del software. Esto evidencia una valiosa lección: la necesidad del mantenimiento y de hacer correcciones posteriores en un proyecto de software se puede minimizar, pero es muy difícil eliminarla del todo. Por otro lado, a nivel más personal, si tuviese que decir un aspecto que me hubiese gustado hacer mejor en este trabajo es la comunicación: por un conjunto de razones personales y académicas, una vez finalizada la parte práctica de este trabajo, no pude escribir la memoria hasta 5 años después; la frustración derivada de este retraso hizo que no comunicase tan bien como me habría gustado la falta de progreso en la redacción de la memoria.

Siguiendo con la valoración a nivel personal, son muchos los aspectos positivos que han surgido de la realización de este proyecto. En primer lugar, ha sido un placer colaborar con el grupo de Eduardo Eyras, me sentí muy a gusto y la experiencia de estar en un centro de investigación como el PRBB fue muy estimulante. Por otro lado, es muy importante la satisfacción de aportar un trabajo que resultó útil a la investigación científica, particularmente en el ámbito del cáncer. Como alguien que en estos

momentos pretende buscar su futuro profesional en el ámbito de la investigación, es un orgullo y un privilegio tener un articulo publicado como coautor en una etapa tan temprana. Además, valoro mucho los conocimientos que adquirí y el proceso de aprendizaje que realicé durante el proyecto. Sin embargo, quizá lo más significativo de este proyecto fue el ayudarme a recuperar mi pasión por la ciencia y la biología en particular, y la experiencia fue decisiva para la decisión del cambio de rumbo de mi carrera académica y profesional: en estos años he finalizado el Grado en Biología y en estos momentos estoy cursando el Master en Genética con la perspectiva de continuar el año que viene con un doctorado y el objetivo de dedicarme profesionalmente a la investigación científica, con mi bagaje de ingeniero informático como una herramienta crucial.

11.1 AGRADECIMIENTOS

La presentación de este Trabajo de Fin de Grado acaba una etapa que ha acabado durando 11 años en la que el centro de mi vida ha sido estudiar y esforzarme para aprender, y en última instancia finalizar con éxito las dos carreras universitarias que he cursado. En particular este trabajo ha tenido unos períodos que no han sido fáciles, pero estoy muy satisfecho de haber conseguido sobreponerme y finalizarlo. Quería agradecer brevemente a todas aquellas personas que me han apoyado y que han hecho posible este último paso.

En primer lugar, al Prof. Eduardo Eyras, el tutor de este trabajo, siempre ha sido un placer trabajar con él y ha sido una fuente de inspiración para todo el camino que he hecho desde que le conocí. También a todo el grupo dirigido por el Dr. Eyras, por haberme hecho sentir muy bienvenido en el PRBB, y en particular al Dr. Endre Sebestyén, ahora *group leader* en la Semmelweis University en Hungría, por confiar en mi trabajo y por incluirme como coautor en el artículo que publicó.

En segundo lugar, al Prof. Xavier Messeguer, cotutor y ahora ponente de este trabajo por parte de la UPC, por haberme propuesto la colaboración con el Prof. Eyras y por tener la paciencia y la voluntad de seguir como tutor de este trabajo después de tantos años.

A la Dra. Montserrat Corominas, por su ayuda y su *feedback* en la elaboración de la parte del anexo que trata de la enfermedad del cáncer.

Al Dr. Jordi Garcia, a mi amigo Guillem Reig, y a mi tía Maite Alvare por leerse el anexo divulgativo y darme su opinión.

A todos los profesores de la FIB, que a lo largo de la carrera me han hecho adquirir todos los conocimientos y habilidades que han hecho posible el éxito de este trabajo.

Y finalmente a toda mi familia y mis amigos que me han apoyado y que han creído en mi incluso cuando no tenía claro si llegaría alguna vez a escribir el presente documento. En especial a mi madre, mi padre y mi hermano.

12. MATERIAL ADICIONAL

En el fichero comprimido que se adjunta a este trabajo se incluye:

- El directorio iso-ktsp, con los ficheros README, LICENSE y CONTACT, que forman parte de la documentación del software, así como los ficheros de código fuente dentro del directorio src/ktsp.
- El ejecutable portable iso-kTSP_v1.0.3.jar.
- Dos ficheros comprimidos con ficheros de entrada de ejemplo: brca_gene_read_pairednormaltumor.txt.gz que contiene datos de expresión de genes para la versión estándar del algoritmo y brca_iso_read_pairednormaltumor.txt.gz con datos de expresión de isoformas, para ejecutar la versión para isoformas del algoritmo (opción -i). Nótese que el tiempo de ejecución de la versión para isoformas con el fichero de ejemplo suele estar en el orden de pocos segundos, pero que requiere un uso de memoria superior a un PC habitual. Por otro lado, la ejecución de la versión de genes con el fichero de ejemplo, además de los requisitos de memoria de la versión para isoformas, tiene tiempos de ejecución muy elevados (unas 6 o 7 horas en un servidor del departamento de genética de la UB). Para pruebas más pequeñas y manejables en PCs habituales se recomienda reducir el tamaño de la entrada eliminando cualquier numero de líneas (exceptuando la primera, que actúa como cabecera).

El código fuente y el ejecutable está también disponible en la web:

https://bitbucket.org/regulatorygenomicsupf/iso-ktsp/src/master/

Todos los ficheros de entrada que se usaron en el proyecto de investigación están disponibles en la web:

https://figshare.com/articles/TCGA_Iso_kTSP_analysis_dataset/1061917

ANEXO: INTRODUCCIÓN A LA BIOLOGÍA, LA GENÉTICA Y LA ENFERMEDAD DEL CÁNCER.

Este anexo se ha escrito teniendo en mente un público sin ningún conocimiento previo en biología celular y genética. El objetivo es proporcionar al lector una explicación informal pero rigurosa de la base científica que sustenta el proyecto y entender el objetivo científico que se pretende abordar con el software que se ha desarrollado y la potencial relevancia de la investigación en la que se enmarca este proyecto. Así, partiendo de los aspectos más básicos de la biología y bioquímica celular, se pretende explicar todo lo necesario para entender qué es y qué función tiene el *splicing* alternativo, y la posible relación de este mecanismo con la enfermedad del cáncer. Aunque es inevitable entrar en cierto detalle para explicar un mecanismo tan específico, se intentará evitar en la medida de lo posible los detalles moleculares y mecanísticos de los procesos de los que se habla, centrándonos más en el aspecto funcional.

Así, se plantea un recorrido desde los conceptos más generales de la biología y la bioquímica de la célula, para luego ir de manera más específica hacia el campo de la genética molecular, viendo los procesos de replicación, mantenimiento y expresión de los genes, hasta llegar al mecanismo de *splicing* alternativo. A continuación, se explicará muy brevemente la enfermedad del cáncer, el cómo y por qué se desarrolla la enfermedad a nivel genético.

LA CÉLULA: ¿DE QUÉ ESTÁ HECHA Y CÓMO FUNCIONA?

Todos los organismos vivos que conocemos están formados por una o más células. La excepción son los virus, que no está claro si se deben clasificar como organismos vivos o no. Una célula, en esencia, es una entidad que ocupa un espacio delimitado por una membrana, y que es capaz de mantener en ese espacio un ambiente químicamente diferenciado respecto al ambiente externo (mediante diferentes mecanismos químicos). Además, en general, las células tienen la capacidad de replicarse, es decir, crear copias más o menos exactas de sí mismas. Las células necesitan obtener energía y componentes químicos del ambiente para mantener la diferencia respecto al medio exterior y para construir y reponer los mecanismos que usan para mantenerla. La diversidad de estrategias para obtener esa energía y componentes junto con las diferentes estrategias para replicarse son la base de la gran diversidad de formas de vida que conocemos.

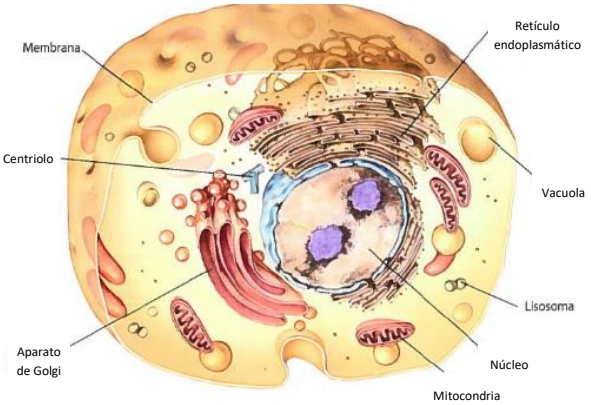

Fig. A1. Esquema general de una célula Eukaryota.
Fuente: Jimeno y Ugedo (2008) [17]

Hay organismos (Bacterias y Arqueas) que estructuralmente responden a la idea más básica de la célula que hemos explicado. Sin embargo, en este documento nos centramos en el ser humano que es estructuralmente más complejo. En primer lugar, formamos parte del grupo de los Eukaryotas, cuya característica principal es la presencia, dentro de la célula, de diversas estructuras y compartimentos membranosos (fig. A1). Para nosotros, la estructura más relevante es el núcleo, que es un espacio dentro de la célula delimitado por su propia membrana. Como iremos viendo con más detalle, el núcleo es el repositorio de información, donde se encuentran todas las instrucciones que puede usar la célula. Todo el espacio de la célula que no es el núcleo se llama citoplasma. Las mitocondrias merecen una mención especial; son estructuras también delimitadas por membrana cuya función principal es la participación en la generación de la energía para el resto de la célula. El ser humano, además de ser Eukaryota, es un organismo multicelular, es decir, estamos compuestos por muchas células que cooperan entre sí y funcionan conjuntamente. Además, en el cuerpo humano hay muchos tipos de células que están especializadas en distintas funciones y se agrupan en tejidos y órganos para realizar sus funciones conjuntamente.

A nivel químico, las células están compuestas por cuatro tipos de componentes, llamados macromoléculas. Hay macromoléculas, llamadas polímeros, que están compuestas por la unión encadenada de moléculas más pequeñas, llamadas monómeros. A continuación, presentamos brevemente los 4 tipos de macromoléculas, su estructura química y su función.

- Lípidos (fig. A2.A): la mayor parte de lípidos están compuestos por uno o más ácidos grasos unidos a otra molécula. Los ácidos grasos son largas cadenas de átomos de carbono con hidrógenos alrededor de la cadena de carbonos. Los lípidos son el componente estructural de la membrana celular y el resto de estructuras membranosas de la célula. Además, en el cuerpo humano funcionan como una reserva energética a largo plazo, sobretodo en tejidos especializados.
- Glúcidos (o hidratos de carbono) (fig A2.B): los glúcidos más simples se llaman monosacáridos. Los monosacáridos también son cadenas de átomos de carbono (como mínimo 3), pero cada carbono de la cadena tiene unido un hidrogeno y un grupo hidroxilo (formado por un oxígeno y un hidrógeno), con una excepción: un carbono que tendrá un oxigeno unido por doble enlace. Muchos de los monosacáridos pueden adoptar una forma circular, y es como se suelen representar. Los monosacáridos pueden unirse unos a otros para formar glúcidos

más complejos: disacáridos, oligosacáridos y polisacáridos. La función más importante de los glúcidos es hacer de reserva energética a corto plazo.

- Proteínas (fig. A2.C): las proteínas son largas cadenas formadas por monómeros llamados aminoácidos. Los aminoácidos tienen una estructura que consiste en un átomo de carbono central que está unido a 4 grupos: un hidrogeno, un grupo carboxilo (un carbono unido a un oxígeno y un grupo hidroxilo), un grupo amino (un nitrógeno unido a dos hidrógenos), y un grupo variable llamado radical. El grupo carboxilo de un aminoácido se une al grupo carboxilo del siguiente para formar las largas cadenas que son las proteínas. En biología principalmente se encuentran 20 aminoácidos que se distinguen por el radical variable, que les da una gran variedad de propiedades químicas distintas. La combinación de los 20 aminoácidos en largas cadenas de hasta decenas de miles de monómeros en algunos casos, con las diferentes interacciones químicas que pueden tener entre ellos, da lugar a que las proteínas sean extremadamente versátiles, tanto a nivel estructural como funcional. Por eso, las proteínas son la base de la diversidad de funciones e interacciones que pueden realizar las células, incluidas reacciones químicas como procesar los alimentos, el movimiento de los músculos o la transmisión de las señales nerviosas. Además, muchas proteínas interaccionan entre sí, cooperando para conseguir realizar funciones más complejas, o regulando las funciones que hacen otras proteínas. Hay que remarcar que lo que define una proteína es la secuencia de aminoácidos que la compone: si cambiamos un aminoácido de la secuencia por otro, la proteína cambia, aunque el cambio puede tener un impacto mayor o menor en la función de la proteína.

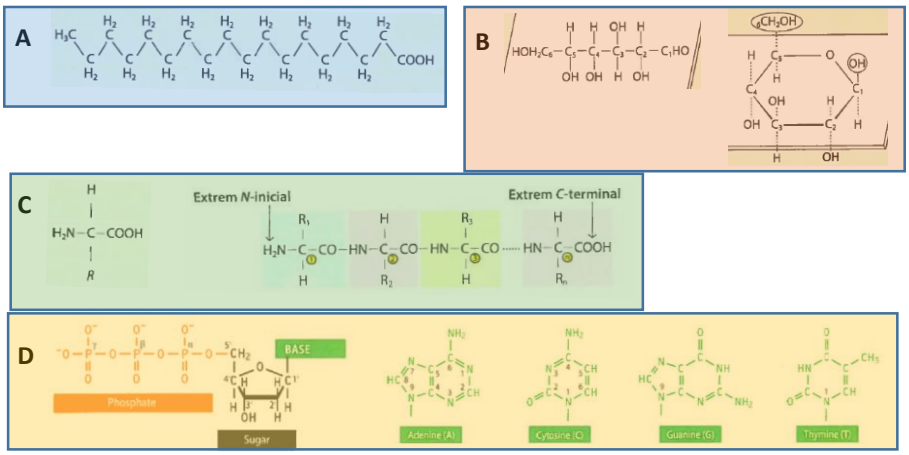

*Fig. A2. Estructura química de las macromoléculas. **A**: Ácido graso, componente principal de los lípidos. **B**: La glucosa, como ejemplo de monosacárido. A la izquierda, la forma lineal. A la derecha, la forma circular. **C**: A la izquierda, la forma general de un aminoácido: R representa al radical variable, que puede ser uno de los 20 grupos químicos distintos. A la derecha, esquema de la unión de los aminoácidos para formar la cadena de la proteína. **D**: A la izquierda, estructura general de un nucleótido, con la pentosa en negro, el grupo fosfato en naranja y la posición de la base nitrogenada en verde. A la derecha, las 4 bases nitrogenadas que forman parte del ADN. En el ARN se encuentra el uracilo en lugar de la timina.*
Composición propia. Fuentes: Jimeno y Ugedo (2008) [17] y Brown (2018) [15]

- Ácidos nucleicos (fig A2.D): están formados por monómeros llamados nucleótidos. Un nucleótido está formado por una pentosa (un glúcido de cinco carbonos), que por un lado tiene unido un grupo de uno a tres fosfatos (un fosforo unido a varios oxígenos) y por el otro tiene un grupo variable, llamado base nitrogenada (porque tienen varios nitrógenos). Uno de los carbonos de la pentosa se puede unir al fosfato del siguiente nucleótido para formar largas cadenas de hasta millones de nucleótidos. Los ácidos nucleicos están formados por 5 tipos de nucleótidos, que se suelen simbolizar con una letra: Adenosina (A), Guanosina (G), Citidina (C), Timidina (T) y Uridina (U). Individualmente, los nucleótidos tienen funciones muy importantes en la célula, como por ejemplo el ATP (adenosina con tres fosfatos), que es la fuente de energía inmediata que usan la mayoría de procesos en la célula. Sin embargo, los ácidos nucleicos tienen la función de codificar toda la información que la célula necesita para funcionar. Hay dos tipos de ácidos nucleicos: el ADN (ácido desoxirribonucleico) y el ARN (ácido ribonucleico). Las funciones e interacciones entre los dos tipos de ácidos nucleicos se explicarán en el próximo apartado, pero las diferencias estructurales son en una ligera diferencia en la pentosa, y en que el ADN tiene timidinas (T) pero no Uridinas (U) y el ARN tiene Us pero no Ts.

Además de estos tipos de moléculas, existe una variedad de moléculas mixtas, como glicoproteínas, glucolípidos, lipopolisacáridos, etc., que también tienen funciones importantes, por ejemplo, en la recepción de estímulos del exterior de la célula y la transmisión de esas señales hacia la célula.

LA GENÉTICA: LA INFORMACIÓN QUE DEFINE UN ORGANISMO Y COMO SE TRANSMITE A LA DESCENDENCIA

En la tierra hay especies muy distintas de organismos, cada una con sus rasgos característicos. Además, en la mayoría de los casos, los individuos de la misma especie no son idénticos, sino que hay una variabilidad intraespecífica más o menos pronunciada. A nivel molecular, tanto las diferencias entre especies como entre individuos de una misma especie se pueden explicar en su mayoría por las diferencias en sus proteínas, ya sea por la presencia o ausencia de proteínas enteras, por pequeñas diferencias en la secuencia de aminoácidos o diferencias en la cantidad de una proteína que se produce en las células. La observación de que un organismo hereda muchos de los rasgos de sus progenitores implica la existencia de algún mecanismo de transmisión de la información sobre las proteínas de un organismo a la descendencia. La genética es el estudio de ese mecanismo de transmisión de información heredable, de cómo se hereda y de cómo se extrae y usa esta información en las células.

Hoy se sabe que la información genética está contenida en el ADN, que se ubica en el núcleo de una célula. El ADN contiene la información sobre las proteínas de un organismo. Sin embargo, sabemos que las proteínas se definen por su secuencia, que está compuesta por 20 aminoácidos distintos. El ADN, en cambio, está compuesto por una secuencia de 4 tipos de nucleótidos distintos: A, G, C y T. Para alguien del ámbito de la informática, la solución es sencilla: agrupando la secuencia de nucleótidos de 3 en 3, tenemos 64 posibles códigos, suficientes para codificar por cada uno de los 20 aminoácidos. Y en efecto, así es como funciona el ADN (tabla A3): la secuencia de aminoácidos de las proteínas está codificada por la secuencia de tripletas de nucleótidos en el ADN. Cada tripleta que codifica para un aminoácido se llama codón. Además, si tenemos 64 códigos para codificar 20 aminoácidos, tenemos muchos códigos que sobran, lo cual permite que un aminoácido pueda tener más de un código asociado. Más adelante veremos la relevancia de esta redundancia.

UUU	Phe	UCU	Ser	UAU	Tyr	UGU	Cys
UUC		UCC		UAC		UGC	
UUA	Leu	UCA		UAA	Stop	UGA	Stop
UUG		UCG		UAG		UGG	Trp
CUU	Leu	CCU	Pro	CAU	His	CGU	Arg
CUC		CCC		CAC		CGC	
CUA		CCA		CAA	Gln	CGA	
CUG		CCG		CAG		CGG	
AUU	Ile	ACU	Thr	AAU	Asn	AGU	Ser
AUC		ACC		AAC		AGC	
AUA		ACA		AAA	Lys	AGA	Arg
AUG	Met	ACG		AAG		AGG	
GUU	Val	GCU	Ala	GAU	Asp	GGU	Gly
GUC		GCC		GAC		GGC	
GUA		GCA		GAA	Glu	GGA	
GUG		GCG		GAG		GGG	

Tabla A3. El código genético. La tabla ilustra la correspondencia entre los codones y los aminoácidos que codifican. Los codones Stop no codifican para ningún aminoácido, sino que indican el final de la traducción. *Fuente: Brown (2018) [15]*

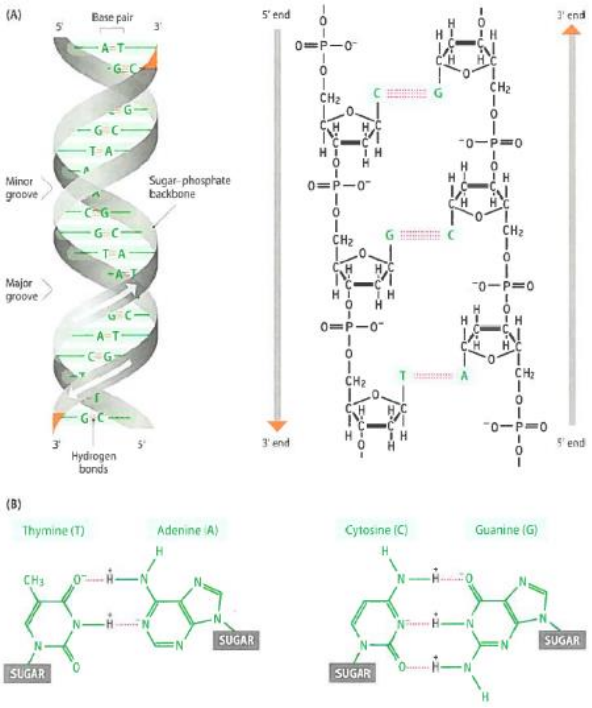

*Fig. A4. **A**: Estructura de la doble cadena del ADN, con las cadenas paralelas compuestas por bases complementarias unidas con los puentes de hidrógeno. **B**: Detalle del funcionamiento de los puentes de hidrógeno en la complementariedad de bases.* *Fuente: Brown (2018) [15]*

Una de las características más relevantes del ADN tal y como se encuentra en las células de los organismos vivos es la estructura de doble cadena (o doble hélice) (fig. A4.A). La idea es que las moléculas de ADN están siempre emparejadas: una cadena siempre está unida en paralelo a otra de la misma longitud. Además, la secuencia de las dos cadenas que están unidas no es independiente. Esto ocurre por un fenómeno llamado complementariedad de bases (fig. A4.B): los nucleótidos de una cadena interaccionan con los de cadena paralela formando unas uniones químicas débiles (llamadas puentes de

hidrógeno) y, en concreto, cada nucleótido tiene preferencia por unirse a su complementario. Las As se unen a las Ts y las Gs a las Cs. Esto quiere decir que, a partir de una de las cadenas, podemos deducir la secuencia de la otra. La complementariedad de bases tiene mucha relevancia para todos los procesos en los que interviene el ADN, como veremos más adelante.

En este punto vale la pena hablar de lo que es un gen. Tradicionalmente, un gen se define como la unidad de herencia: un rasgo que se hereda esta codificado por un gen. En realidad, la cosa es más complicada. Hay rasgos que no se heredan en bloque, porque están influenciados por más de un gen, y hay genes que influencian varios rasgos. A nivel molecular, un gen es una región continua de la secuencia de ADN que potencialmente puede tener un producto a nivel molecular, es decir, que codifica para una molécula concreta. Cuando se produce la molécula que esta codificada por un gen, se dice que ese gen se está expresando. En el siguiente apartado, hablaremos de algunos detalles del proceso de expresión génica, que son centrales para el proyecto que se describe en esta memoria. En este trabajo, nos centraremos en los genes que codifican para proteínas, es decir, cuyo producto final de la expresión de ese gen es una proteína. Sin embargo, no todos los genes codifican para proteínas, en cuyo caso el producto final de la expresión es una molécula de ARN, que en muchos casos tiene funciones muy diversas, aunque las funciones de muchos genes no codificantes no están tan bien estudiadas como las de los codificantes.

En el ser humano, como en todos los organismos multicelulares, la codificación de la información genética en la secuencia de ADN tiene varias particularidades. En primer lugar, la información genética no está en una sola molécula continua de ADN, sino que está en varias moléculas separadas, llamadas cromosomas, cada cromosoma formado por las dos cadenas complementarias que mencionamos antes. Cada cromosoma contiene un conjunto de genes propios, que generalmente no comparte con otros cromosomas. Hay que mencionar brevemente que además de los cromosomas que están en el núcleo, las mitocondrias tienen su propio cromosoma con sus propios genes, sin embargo, nosotros nos centraremos en el ADN nuclear. En la especie humana, los 3,2 millones de nucleótidos (solo contando una de las cadenas complementarias) que forman el ADN nuclear están repartidos en 24 cromosomas diferentes. Además, con la excepción de los dos cromosomas sexuales, tenemos dos copias de los 22 cromosomas restantes, una copia que heredamos de cada uno de nuestros progenitores. Esto añade un nivel de redundancia y de diversidad a la información genética. Tenemos dos copias de la mayoría de genes, que pueden ser copias idénticas o no, en función de las variantes de los genes que tenían los progenitores. Otra característica importante es que, en principio, todas las células del cuerpo humano tienen el mismo contenido genético, el genoma al completo. Sin embargo, distintos tipos de células usarán distintos genes, porque tienen funciones especializadas, por lo tanto, aunque una célula tenga la información de todos los genes, solo expresará aquellos que necesite. Finalmente, una

última particularidad del ADN humano es que del total de 3,2 millones de nucleótidos que forma las secuencias de los cromosomas, solamente entre un 1% y un 2% son genes que codifican para proteínas. Se sabe que al menos una parte del ADN restante tiene una función reguladora: ayuda a establecer cuando y en qué células se expresan los genes codificantes. No se sabe con certeza que porcentaje del ADN no codificante tiene alguna función, ya que constantemente se están descubriendo nuevos mecanismos por los que las secuencias hacen funciones alternativas a la de codificar proteínas. Sin embargo, el hecho de que un porcentaje tan pequeño del ADN sea codificante es un indicio de la complejidad del funcionamiento del genoma humano más allá de la codificación de proteínas.

LA EXPRESIÓN GÉNICA, EL ARN Y EL *SPLICING* ALTERNATIVO

Hemos visto que la información codificada en los genes tiene que expresarse para hacer su función, es decir, en el caso de los genes codificantes, se tienen que producir las proteínas que estos codifican. Hemos visto también que el hecho de que las células de los organismos multicelulares estén especializadas significa que necesitarán hacer diferentes funciones, en principio a través de diferentes proteínas. Además, la cantidad y los tipos de proteínas que se necesitan varían en el tiempo, ya sea en función de señales externas o del estado interno de la célula. Esto quiere decir que hay un proceso de regulación genética, generalmente bastante complejo, que decide qué genes se expresarán en cada momento y célula determinados y en qué cantidad.

La regulación de la expresión es un tema muy complejo y aquí no entraremos en detalles, sin embargo, cuando veamos el proceso general de la expresión génica es importante tener en cuenta que cada paso que veremos tiene sus mecanismos de regulación. Así, antes de entrar en la expresión, creo interesante poner un ejemplo muy sencillo de regulación. Para ello, hay que tener en cuenta que la propia expresión de genes se basa en un complejo de proteínas que interacciona con el ADN, leyendo la información codificada, y que, a través de varios pasos, construye la proteína que esta codificada. Ahora supongamos un nutriente que requiere de un mecanismo de varias proteínas para ser procesado y aprovechado por la célula. Como producir la maquinaria proteica para procesar el nutriente requiere un gasto para la célula, tiene sentido que solo se produzca la maquinaria cuando el nutriente esté presente en la célula. Para ello, hay una proteína receptora (que siempre se está produciendo en bajas cantidades) que puede detectar el nutriente en la célula. Cuando la proteína receptora interactúa con el nutriente, se activa y va al núcleo, uniéndose físicamente al ADN en las posiciones donde están los genes del mecanismo que procesa el nutriente. Una vez unida al ADN, esta proteína receptora atrae a las proteínas del mecanismo que se encarga de expresar los genes hacia las posiciones de los genes que se van a expresar, y empieza el proceso de expresión génica.

Veamos ahora con algo más de detalle los pasos de la expresión génica, es decir, como se transforma la información codificada en la secuencia de ADN del gen a la secuencia de aminoácidos de la proteína correspondiente. El aspecto más importante del proceso de expresión es que, a grandes rasgos, se hace en dos pasos y hay una molécula intermedia. Esta molécula es el ARN. De hecho, cuando se expresa cualquier gen se produce ARN, tanto si codifica para proteínas como si no. En los genes que no codifican para proteínas, el ARN suele ser el efector que hace la función del gen. En el caso de los genes codificantes, el ARN es un intermediario que porta la información del gen y que se lee para producir la proteína; en este caso, a la molécula de ARN se la llama ARN mensajero (ARNm). La producción del ARN a partir del ADN se llama transcripción y el paso del ARN a proteína se llama traducción. Cuando un gen se expresa, se suelen producir muchas copias de la molécula del ARN y de cada molécula del ARN se producen muchas copias de la proteína. La molécula de ARN que se produce en la transcripción es una copia de la secuencia del gen que está en el ADN, pero en el caso de la especie humana, el ARN pasa por varios procesos de maduración antes de producir la proteína. Si imaginamos la célula como un sistema informático, el ADN sería un repositorio con el código de todos los programas que puede ejecutar el sistema, las moléculas de ARN serían copias locales de los programas que se van a ejecutar en ese momento, y el proceso de maduración y traducción a proteína sería análogo a compilar y linkar los programas para ejecutarse.

El primer paso de la expresión de un gen es la transcripción (fig. A5), o la creación de una molécula de ARN a partir de un gen de ADN. La transcripción es una copia directa de la secuencia del ADN correspondiente al gen transcrito. El proceso se basa en la complementariedad de bases. La transcripción la realiza un complejo de proteínas, la más importante de las cuales es la ARN-Polimerasa. Hay secuencias específicas del ADN al principio del gen a las que se une la ARN-Polimerasa y el resto de proteínas del complejo, y la transcripción empieza a partir de un punto cercano a ese punto de unión. El complejo recorre la secuencia de ADN, separando las dos cadenas de ADN a su paso y volviendo a juntarlas una vez ha pasado. En la zona abierta actúa la ARN-polimerasa, leyendo uno a uno los nucleótidos de una de las secuencias y seleccionando el nucleótido complementario para añadirlo a la

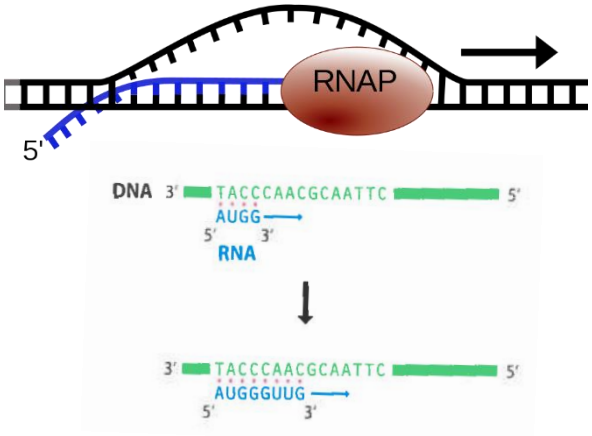

Fig. A5. Esquema del proceso de transcripción. Arriba, en negro, las dos cadenas de ADN que se abren al paso de la ARN-Polimerasa; en azul, la cadena de ARN que se está sintetizando, complementaria a una de las cadenas de ADN. Abajo, la secuencia de ADN en verde y la secuencia de ARN que se está creando en azul. *Composición propia, fuentes: Brown (2018) [15] y Wikimedia commons, dominio público, https://commons.wikimedia.org/w/index.php?curid=2892073.*

molécula de ARN que se está creando. Por lo tanto, se está creando una molécula de ARN con la secuencia complementaria a la que se está leyendo, pero que a su vez es una copia de la otra cadena de ADN. Hay que tener en cuenta, sin embargo, que en el ARN no hay el nucleótido T, pero el nucleótido U también tiene complementariedad con la A, con lo cual en la cadena de ARN se sustituyen las Ts del ADN por Us. La transcripción acaba cuando una secuencia especial hace que el complejo de transcripción se desenganche de la cadena de ADN. Hay que tener en cuenta que la ARN-polimerasa no es infalible, y que hay veces que pone un nucleótido que no toca por complementariedad de bases, lo cual provocará que la molécula de ARN no sea una copia exacta del gen de ADN. Sin embargo, estos errores no tienen un impacto importante en la célula: cuando se expresa un gen se transcriben muchas copias de ARN de ese gen, y si una entre muchas contiene un error, no suele haber consecuencias. Además, el hecho de que más de un codón codifique para el mismo aminoácido también ayuda a minimizar el impacto de los errores.

Si el gen es codificante, el proceso de expresión acaba con la traducción (fig. A6): la construcción de la proteína a partir del ARNm maduro. La idea de la traducción es parecida a la transcripción, se lee la secuencia del ARN para ir construyendo la secuencia de aminoácidos de la proteína. En el caso de la traducción, se lee la secuencia por codones, es decir, de tres en tres, y la maquinaria que realiza el proceso es un complejo compuesto por proteínas y unas moléculas especiales de ARN, llamadas ARN ribosomal (ARNr). El ribosoma se une al ARNm y empieza la transcripción a partir de un codón concreto, que a su vez codifica por el aminoácido metionina (con lo cual, en principio, todas las proteínas tienen una metionina como primer aminoácido, aunque posteriormente se puede eliminar). En la traducción también se usa la complementariedad de bases. Existen unas secuencias de ARN cortas, llamadas ARN de

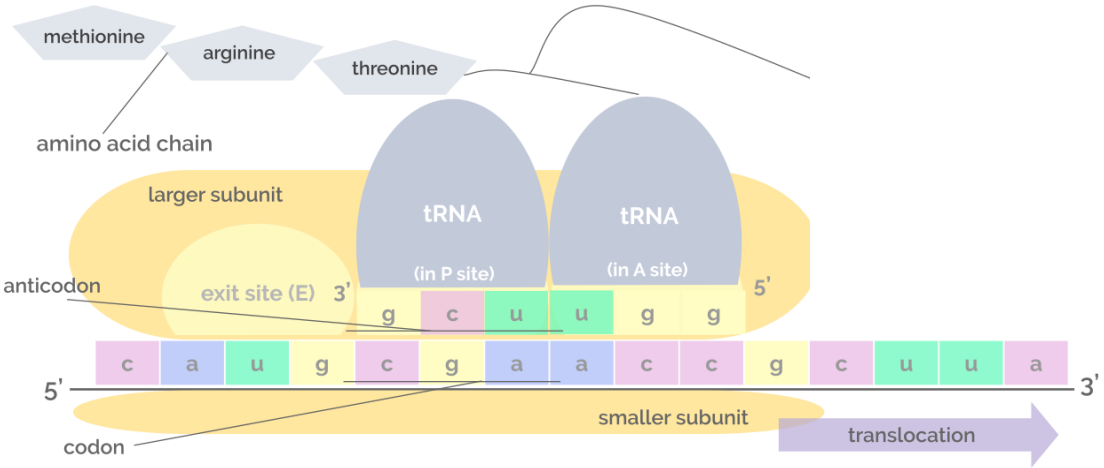

Fig. A6. Esquema del proceso de traducción. En amarillo oscuro, los diferentes componentes del ribosoma. Abajo, la secuencia del ARNm. En gris oscuro, los ARNt con las secuencias complementarias a los codones. En gris claro, la cadena de aminoácidos que está formando la proteína, a partir de los aminoácidos unidos a los ARNt, Fuente: Wikimedia commons, por Jordan Nguyen, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=79164021.

transferencia (ARNt), que se unen a aminoácidos concretos y a su vez tienen una parte muy localizada que contiene la secuencia complementaria a un codón concreto. Así, el ribosoma, al leer un codón del ARNm, atrae un ARNt con su aminoácido e intenta unirlo por complementariedad de bases al codón. Si el ARNt se une correctamente, el ribosoma añade el aminoácido que estaba unido al ARNt a la secuencia de aminoácidos. El ribosoma recorre el ARNm y va creando la secuencia de aminoácidos hasta que se encuentra uno de los codones que indican el fin de la transcripción, momento en que el ribosoma se libera y la proteína está completa. De manera parecida a la transcripción, en una molécula de ARN puede haber varios ribosomas produciendo la proteína en paralelo.

Hay dos particularidades de los Eukaryotas, incluidos los humanos, con respecto a la expresión génica. La primera es que la transcripción y la traducción ocurren en espacios distintos de la célula: la transcripción se realiza en el núcleo, donde está el ADN, mientras que la traducción ocurre fuera del núcleo, en el citoplasma, donde están los ribosomas. Esto quiere decir que el ARNm debe viajar desde el núcleo hasta el citoplasma para completar el proceso de expresión del gen. La segunda particularidad, como ya hemos mencionado, es que el ARN pasa por varios procesos de maduración en los cuales se llega a modificar su secuencia respecto a la original, por lo tanto, la secuencia que se acaba traduciendo no es una copia exacta de la secuencia del gen tal y como se encuentra en el ADN.

De todas las modificaciones que sufre el ARNm en su maduración, nos centraremos en la más espectacular: el *splicing* (fig. A7.A). Si miramos la secuencia original del ARN, o del ADN del cual se copia, veremos que no todo el ARN codifica para la proteína final. En primer lugar, los extremos del ARN no se traducen: hay una región de la secuencia antes del codón de inicio de traducción y otra región después del codón de final de traducción que no codifican para la proteína. Sin embargo, la secuencia que codifica para la proteína no se encuentra toda de manera ininterrumpida. Las regiones codificantes, llamadas exones, están separadas por regiones no codificantes llamadas intrones. El *splicing* es el proceso de cortar los intrones y juntar los exones para acabar con la secuencia codificante continua que se acabará traduciendo. La mecánica del *splicing* es compleja y no la explicaremos en detalle aquí, pero el reconocimiento de los sitios de corte está parcialmente basado en la secuencia que hay alrededor del punto de corte.

Todos los genes codificantes en el ser humano pasan por el proceso de *splicing*. Sin embargo, en muchos casos encontramos una variante del proceso llamado *splicing* alternativo (fig. A7.B). En muchos genes, el proceso de *splicing* no da siempre el mismo

resultado: puede haber exones que se puedan incluir o no en la secuencia final, o incluso exones que son alternativas excluyentes (si se incluye un exón, no se incluirá otro). Este proceso resulta en que a partir de un gen puedan formarse distintos ARNm maduros a partir de distintas combinaciones de exones, y potencialmente, se puedan expresar proteínas distintas a partir de un mismo gen. Las distintas variantes de ARNm que se pueden obtener a partir de un mismo gen se denominan isoformas de ese gen. El *splicing* alternativo y las distintas isoformas que produce aumentan la versatilidad del genoma: el ADN humano solo tiene unos 20.000 genes codificantes, sin embargo, la cantidad de proteínas distintas que se conocen en todo el cuerpo humano es considerablemente mayor. Resulta lógico que el *splicing* alternativo tenga mecanismos que lo regulen, para obtener las isoformas adecuadas en el momento y lugar que interesen y en las cantidades requeridas. Estos mecanismos de regulación del *splicing* alternativo se conocen de una manera más limitada que los mecanismos globales de regulación de la expresión génica, así que tampoco entraremos en ellos, pero es crucial para este trabajo entender que existen y que son complejos. El *splicing* alternativo se podría asimilar a la compilación condicional en lenguajes como C, donde fragmentos de código se pueden incluir o no en función del valor de unos parámetros en el momento de la compilación.

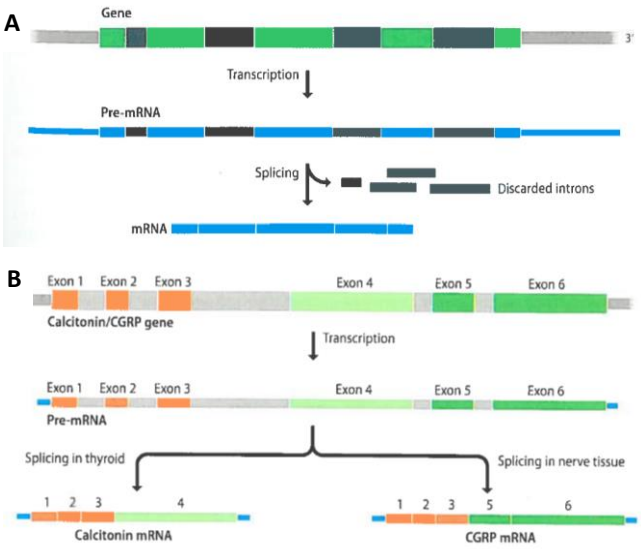

Fig. A7. ***A****: Esquema del proceso de splicing. Arriba, el gen en el ADN, con los exones en verde y los intrones en gris oscuro. En medio, la molécula de ARN inmadura, después de la transcripción, con los exones en azul. Abajo, la molécula de ARNm después del proceso de splicing, donde solo quedan los exones.* ***B****: Esquema del splicing alternativo. En este ejemplo los exones 1, 2 y 3 siempre se incluyen, pero en una isoforma se incluye el exón 4, mientras que en la otra se incluyen los exones 5 y 6.* Fuente: Brown (2018) [15]

LA REPLICACIÓN DEL ADN Y LAS MUTACIONES

Hemos visto ya que una de las características de los organismos vivos es la capacidad de producir descendencia: más individuos de la misma especie. En el caso de los organismos unicelulares, esto se consigue mediante la división celular: una célula crea una copia de sí misma, que ya es un individuo separado. En los organismos multicelulares, como los humanos, la creación de descendencia es más compleja, pero la división celular sigue teniendo un papel fundamental. En los órganos sexuales tenemos células especializadas que se dividen, creando las llamadas células reproductoras (óvulos y espermatozoides) que al unirse con una célula reproductora del

sexo opuesto crea la primera célula del nuevo individuo. Durante el proceso de desarrollo, se crean todas las células del organismo adulto a partir de sucesivas divisiones de esa primera célula. Además, en el individuo adulto, la mayoría de células no duran toda la vida, y tienen que irse reponiendo, por lo tanto, existen células especializadas en dividirse para reponer las células a medida que se necesitan.

En el ser humano, hay dos tipos de divisiones celulares: la meiosis, que se usa específicamente para crear las células reproductoras, y la mitosis, que da lugar a todo el resto de células. La meiosis es más compleja y tiene más pasos que la mitosis, pero ambas tienen en común los pasos básicos: en primer lugar, se crea una copia del ADN de todos los cromosomas, se separan las copias de los cromosomas a lados opuestos de la célula, y finalmente la célula se contrae por el centro, hasta dividirse en dos, cada una con una copia de los cromosomas. Recordemos que cada célula tiene dos copias de cada cromosoma, provenientes de cada uno de los progenitores. En la división, se duplican ambas copias de cada cromosoma, de manera que cada célula sigue teniendo una copia de cada progenitor. La particularidad de la meiosis es que hay una segunda división celular, sin duplicar el ADN, de manera que el resultado son 4 células reproductoras, cada una con una sola copia de cada cromosoma (por lo tanto, con la mitad de ADN que el resto de células). Cuando las dos células reproductoras de sexos opuestos se combinan para crear un nuevo individuo, la célula resultante vuelve a tener las dos copias de cada cromosoma, y de esta manera se obtiene una copia de cada uno de los progenitores.

Veamos ahora el proceso de duplicación o replicación del ADN (fig. A8). La base del proceso es similar al de la transcripción. Hay un complejo de proteínas que se encarga de replicar el ADN, el componente más importante de este complejo es la ADN-Polimerasa. El complejo se une a puntos de la cadena doble de ADN y la separa, pero esta vez, las dos cadenas originales de ADN no se vuelven a juntar. La ADN-polimerasa va recorriendo las dos cadenas de ADN que se han separado y va construyendo las nuevas cadenas, añadiendo los nucleótidos por complementariedad de bases. De esta manera, a partir de las dos cadenas originales, que eran

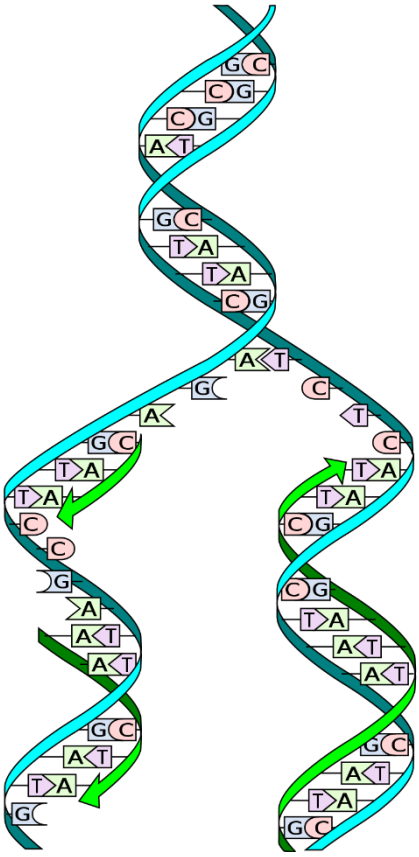

Fig. A8. Esquema del proceso de replicación del ADN. En Azul, las dos cadenas originales. La doble cadena se abre para sintetizar nuevas cadenas, en verde, complementarias a las dos originales. *Fuente: Wikimedia commons, por I, Madprime, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=2497221*

complementarias, se crean dos cadenas nuevas, cada una complementaria a una de las originales.

Al igual que la ARN-polimerasa en el proceso de transcripción, la ADN-polimerasa cuando replica el ADN a veces comete fallos e introduce nucleótidos que no son los que tocaría por complementariedad de bases. Los fallos en la replicación tienen consecuencias mucho mayores que en el caso de la transcripción, ya que solo se está creando una copia de esa cadena: cuando se exprese un gen que tiene un error, todos los productos de ese gen contendrán el error. Además, si la célula que contiene el error a su vez se divide, toda su descendencia tendrá ese mismo error. Por eso, la ADN-polimerasa tiene mecanismos más potentes para detectar y corregir los errores ocurridos al replicar el ADN, sin embargo, hacen de media un error cada 10^7 nucleótidos y algunos de esos errores no se llegan a corregir, de manera que de vez en cuando, una de las células descendientes tendrá en su ADN diferencias respecto al ADN original de la célula progenitora. Estos cambios en la secuencia del ADN respecto a la secuencia original se llaman mutaciones.

Los errores en la replicación del ADN no son el único origen de las mutaciones. Durante la vida de la célula se pueden producir daños en el ADN, de manera intrínseca (el ADN es una molécula muy larga y en ocasiones puede romperse) o por factores externos. Algunos de esos daños son roturas de la cadena, mientras que otros son daños o modificaciones químicas en los nucleótidos. La célula tiene mecanismos para reparar estos daños, aunque no siempre se consigue restaurar la secuencia original, así que también así se producen mutaciones. Hay substancias y factores ambientales que tienen especial propensión a causar daños en el ADN, como los rayos X, o los llamados mutágenos químicos.

Uno de los mecanismos de reparación del ADN es la recombinación homóloga, que usa las distintas copias de un cromosoma que están presentes en el núcleo para intercambiar las cadenas de regiones muy parecidas de dos secuencias. La recombinación homóloga permite el intercambio de fragmentos largos de ADN entre regiones parecidas de dos cromosomas. Esto también ocurre durante la meiosis, y el resultado es que los cromosomas heredados por la descendencia pueden ser una mezcla de las dos copias de los cromosomas originales. Sin embargo, el proceso de recombinación homóloga puede producirse de manera errónea, lo cual da lugar a las mutaciones más extensas: las reorganizaciones cromosómicas, en las que faltan, sobran o se repiten grandes regiones del cromosoma original.

El efecto de las mutaciones puede ser muy variable y depende de la región del ADN donde ocurra. Recordemos que una gran parte del genoma humano no es codificante, y que es posible que algunas regiones no tengan ninguna función. Una mutación en una de esas regiones no producirá ningún efecto en la célula. Si hay una mutación en una región no codificante con función, esa mutación podrá tener un efecto importante, o

casi no tenerlo, en función de la importancia del punto concreto de la mutación para la función del gen. Lo mismo pasa en regiones codificantes: si hay una mutación que cambia la secuencia de aminoácidos de la proteína, el efecto de esa mutación dependerá de la importancia de los aminoácidos afectados para la función de la proteína. Y una vez más se presenta la ventaja de que haya más de un codón codificando para un aminoácido concreto: permite cierta tolerancia a las mutaciones sin que haya cambios de aminoácidos. Otra cosa a considerar es el hecho de que tenemos dos copias de los cromosomas, con lo que, si un gen tiene una mutación en una de las copias, pero la otra no tiene la mutación, en ocasiones la copia buena es suficiente para hacer la función y la mutación no tiene efecto. Si una mutación tiene un efecto, normalmente será negativo: la proteína o el ARN afectado no podrán hacer su función o no la harán de una manera tan eficaz. Sin embargo, en raras ocasiones, el cambio será ventajoso: la proteína mutante hará su función de manera más eficiente o incluso podrá hacer una función que no hacía hasta ahora.

Hay que tener en cuenta también que el efecto de las mutaciones sobre un organismo depende de la célula en la que ocurra la mutación: las mutaciones se propagan cuando se replica el ADN en la división celular. Si una mutación ocurre en una célula que no se va a dividir más, el efecto será mínimo: es una sola célula que funciona diferente entre miles de células. Una mutación en una célula que se divida para reponer las células de un tejido afectará a todas las células de ese tejido que desciendan de la célula mutada. Las mutaciones que tienen un impacto mayor son aquellas que se producen en las células reproductoras: todas las células de un organismo nuevo descienden de la combinación de dos células reproductoras, por lo tanto, una mutación en una de ellas se propagará a todo el organismo. De hecho, las mutaciones son el origen de todas las diferencias genéticas que hay en los seres vivos, tanto entre especies como entre individuos de la misma especie: cambios que se han ido acumulando a lo largo de miles de millones de años y que dan la diversidad actual de la vida en la tierra.

LA ENFERMEDAD DEL CÁNCER

El cáncer es hoy en día una de las enfermedades que más interés y esfuerzo de investigación suscita en los países desarrollados. Esto se debe a que es una de las pocas enfermedades graves (y muchas veces mortal) que, afectando a un número elevado de personas, aún no hemos sabido controlar de una manera eficaz mediante la medicina actual. Además, es una enfermedad que cobra prevalencia con la esperanza de vida de la población: cuanto más avanzada es la edad, más probabilidades de desarrollar cáncer.

La enfermedad del cáncer tiene la particularidad de que no está causada por un agente infeccioso, sino que se trata de un proceso intrínseco al organismo. En concreto, la enfermedad se desarrolla cuando algunas células del organismo se rebelan contra el propio organismo, escapando a su control y expandiéndose hasta afectar las funciones normales del organismo. El origen del cáncer es la acumulación de mutaciones en las

células. Sin embargo, para entender conceptualmente el cáncer, antes debemos comentar un aspecto muy importante del funcionamiento normal del organismo. Como ya hemos comentado antes, en los organismos multicelulares, las distintas células cooperan y se especializan, organizadas en tejidos y órganos, para aportar distintas funciones al organismo. Para que el engranaje en su conjunto funcione, las células tienen que estar organizadas espacialmente según su rol, con lo que el número y la disposición de las células debe ser el adecuado.

Para mantener esa estructura y organización de las células es necesario un mecanismo de control sobre la duración de la vida de las células, su división y su muerte. La mayoría de las células del organismo tienen un tiempo de vida útil determinado (por desgaste externo o interno), tras el que deben morir y ser reemplazadas. Por lo tanto, las células encargadas de dividirse para reponer el resto de las células deben hacerlo de manera controlada, tanto en el tiempo como en la intensidad de la división. Las células de la piel, por ejemplo, tienen un desgaste externo muy importante y, por lo tanto, de por sí, su tasa de reposición será más elevada que en otros tejidos, sin embargo, aún deberá aumentar más si nos hacemos una herida.

Los mecanismos de control de estos y muchos otros procesos celulares pueden involucrar a decenas de proteínas y genes, y se pueden conceptualizar en vías de señalización, que son procesos compuestos por distintos pasos (que involucran distintas moléculas) en los que el resultado de un paso es necesario para que se realice el siguiente. Por ejemplo: una proteína A detecta una sustancia tóxica uniéndose a ella, la proteína A a su vez activa una proteína B que se encarga de activar la expresión del gen C, produciéndose la proteína C que actúa para protegerse de la sustancia tóxica. Este ejemplo de vía es muy sencillo, sin embargo, lo que se pretende destacar es que una mutación en cualquiera de las tres proteínas que impida alguno de los pasos tendrá el mismo efecto: una falta de respuesta protectora frente a la sustancia tóxica. Un nivel mayor de complejidad respecto a las vías son las redes: se habla de redes reguladoras cuando las interacciones para el control de la expresión no son lineales.

Así, el cáncer se produce cuando ocurren una serie de mutaciones que hacen que fallen los mecanismos de control de la división y la muerte celular. Un grupo de células empiezan a dividirse sin control, el crecimiento descontrolado de las células crea deformidades en el tejido u órgano llamadas tumores, que pueden acabar invadiendo el espacio de las células sanas, impidiendo que hagan su función. Las células afectadas por el cáncer pueden incluso viajar a través del organismo para crear tumores secundarios, en el proceso llamado metástasis. Hay que tener en cuenta, sin embargo, que el desarrollo del cáncer y sus efectos malignos es progresivo, a través de sucesivas mutaciones. En los estudios científicos se han perfilado una serie de características que adquieren las células a lo largo del progreso de la enfermedad del cáncer. A continuación, explicamos brevemente estas características:

- Activación desregulada de la división celular: una serie de vías que integran las señales, producidas tanto en el interior de la célula como provenientes de su exterior (incluidas señales de otras células) regulan cuando debe activarse la división celular y, en consecuencia, regulan la expresión de todos los mecanismos implicados. Si hay mutaciones que hagan que la célula active la división celular de manera independiente a las señales que la regulan, la célula empezará a dividirse descontroladamente, y transmitirá esas mutaciones a sus descendientes, cosa que, si no se frena, acabará con un crecimiento exponencial del número de células mutadas.
- Resistencia a la muerte celular: las células sanas tienen un mecanismo de muerte controlada llamado apoptosis. Cuando una célula ha llegado al fin de su vida útil o cuando detecta que está en un estado excesivamente malo (por ejemplo, porque detecta demasiado daño en su ADN), se activa este mecanismo. La apoptosis también se puede activar mediante señales externas, por ejemplo, el sistema inmune puede detectar que una célula está en mal estado (por una infección vírica o por mutaciones) y enviar señales que activen la muerte de esa célula. Mutaciones en la vía de la apoptosis que dan lugar a que no se active la muerte programada son un requisito para continuar proliferando a pesar de las mutaciones que se van acumulando.
- Adquisición de la inmortalidad replicativa: las células sanas tienen unas limitaciones intrínsecas al número de divisiones sucesivas que pueden llegar a realizar. Existen células especiales que tienen mecanismos para superar esta limitación (en los órganos sexuales), pero en el resto de las células estos mecanismos no están activos. Las células que desarrollan cáncer acaban activando de forma anómala estos mecanismos, de manera que pueden dividirse de manera indefinida, creando líneas celulares inmortales.
- Activar la angiogénesis: todas las células del organismo necesitan acceso a los nutrientes y al oxígeno para la producción de la energía y los componentes moleculares que necesitan para funcionar. La distribución de nutrientes y oxígeno se hace a través del sistema vascular (venas y arterias), que se crea durante el desarrollo para llegar a todas las células del organismo, pero la creación de nuevas venas y arterias está muy limitada en el organismo adulto. Las células tumorales también necesitan acceso a los nutrientes y al oxígeno, sin embargo, al crecer de manera descontrolada, pierden el contacto con el sistema vascular, cosa que limita el potencial de crecimiento del tumor. Para poder continuar creciendo, las células tumorales deben adquirir la capacidad de expresar las señales que estimulan la creación de nuevos vasos. Este es un caso que ilustra la complejidad del cáncer: las células afectadas por las mutaciones son capaces de cambiar el comportamiento de células genéticamente sanas a través de señales que normalmente no recibirían. Esto quiere decir que las interacciones en el tumor son más complejas y para entender el funcionamiento global no basta en fijarse en las células afectadas por las mutaciones, sino también como interactúan para su beneficio con las células normales.

- Capacidad de invasión y metástasis: un tumor puede tener consecuencias limitadas si simplemente se desarrolla ocupando un espacio libre de otras células (tumor benigno). Sin embargo, las consecuencias son más graves si las células desarrollan la capacidad de invadir el espacio que ocupan las células sanas, interfiriendo así de manera mucho más agresiva en la función que debería realizar el tejido o órgano. El caso más extremo de desarrollo del cáncer es aquel en que hay células tumorales que adquieren la capacidad de separarse del tumor, de pasar a la circulación y de colonizar otros tejidos en partes distantes del organismo. Es la metástasis, que causa una proliferación de tumores secundarios en todas las localidades donde las células tumorales se establecen con éxito.
- Evasión del sistema inmune: el organismo tiene un sistema compuesto de células especializadas para defenderse de amenazas: el sistema inmune. Algunas de las células del sistema inmune están especializadas en detectar células defectuosas del propio organismo y eliminarlas, ya sea causando la apoptosis o por otros medios. Es habitual que las células tumorales adquieran la capacidad de no ser detectadas o afectadas por el sistema inmune, aunque no está claro si esto es un requisito indispensable para la progresión de la enfermedad.

Todas las características que acabamos de explicar derivan de mecanismos que ya están presentes en el organismo, y que en momentos determinados son utilizados de manera normal por células concretas. Lo que ocurre en el cáncer es que las mutaciones acumuladas afectan a la regulación de esos mecanismos: todas las células al tener el mismo ADN tienen las instrucciones necesarias para ejecutar esos mecanismos, lo que falla es el sistema de control. La complejidad del cáncer se deriva en parte en que fallos en distintos puntos de las vías de control pueden dar lugar al mismo resultado. Sin embargo, hay que tener en cuenta que para la mayoría de estos procesos cruciales existen vías de control redundantes, y para que el cáncer progrese, deben haberse acumulado suficientes mutaciones para afectar a muchas de las vías redundantes.

Un concepto importante es que las mutaciones se van acumulando a lo largo del progreso de la enfermedad. Como hemos visto, las mutaciones pueden aparecer como errores en la copia del ADN durante la división celular, de manera que si se descontrola la tasa de división, se acelera el proceso de adquisición de nuevas mutaciones, acelerando el progreso de la enfermedad y aumentando la posibilidad de que aparezcan las siguientes mutaciones que den el resto de las características que hemos visto. Es habitual en tumores que aparezcan mutaciones que afecten a los mecanismos de reparación y corrección de errores en el ADN, porque cuando aparecen, aceleran aún más la aparición de nuevas mutaciones, aumentando a su vez las probabilidades de adquirir las características que hacen avanzar la enfermedad. En este sentido, existe la diferenciación conceptual de las mutaciones *driver* (conductoras) y de las *passenger* (acompañantes). Las primeras son las causantes de la progresión de la enfermedad, mientras que las segundas pueden beneficiar o acelerar el progreso y, por lo tanto, ser

habituales en el cáncer, pero sin ser las causas directas del progreso del de la enfermedad. Aunque en la práctica puede resultar difícil establecer en muchos casos cuales son las mutaciones *driver* y cuales *passenger*, disponer de la secuencia del ADN de muchos tumores es una buena herramienta para poder hacer una distinción correcta.

Para acabar, comentar que una parte importante de la dificultad de tratar de manera eficaz el cáncer consiste en que las células tumorales son células del propio organismo, aunque ya no respondan al control normal de éste. El resultado es que es difícil afectar, matar o frenar las células tumorales sin afectar excesivamente a las células sanas. Entender cómo funcionan a nivel molecular los mecanismos de control que están afectados de manera consistente en los distintos tipos de cáncer está ayudando a hacer terapias cada vez más eficaces. Sin embargo, la complejidad de esos mecanismos y de las interacciones entre ellos hace que aún tengamos una visión limitada. En los apartados anteriores hemos visto solo una pequeña parte de la complejidad del proceso de expresión de genes, del cual el *splicing* alternativo es una parte importante. De aquí el interés de estudiar qué cambios se observan en la regulación del *splicing* alternativo en la enfermedad del cáncer.

BIBLIOGRAFÍA

REFERENCIAS DIRECTAS

1. BBC Academy. (n.d.). *The history of machine learning*. [online] Disponible en: https://www.bbc.com/timelines/zypd97h [Consulta 10 Sep. 2019].
2. Baldi, P. y Brunak, S. (1998). *Bioinformatics*. 1st ed. Cambridge: MIT Press.
3. Tan, A. y Gilbert, D. (2003). Machine Learning and its Application to Bioinformatics: An Overview. *Bioinformatics Research Centre, Department of Computing, University of Glasgow*. [online] Disponible en: https://www.researchgate.net/publication/2521627_Machine_Learning_and_its_Application_to_Bioinformatics_An_Overview [Consulta 13 Sep. 2019].
4. Olson, R., La Cava, W., Mustahsan, Z., Varik, A. y Moore, J. (2018). Data-driven advice for applying machine learning to bioinformatics problems. *Pac Symp Biocomput*. [online] Disponible en: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5890912/ [Consulta 13 Sep. 2019].
5. Tan, A., Naiman, D., Xu, L., Winslow, R. y Geman, D. (2005). Simple decision rules for classifying human cancers from gene expression profiles. *Bioinformatics*, 21(20), pp.3896-3904.
6. Sebestyén, E., Zawisza, M. y Eyras, E. (2015). Detection of recurrent alternative splicing switches in tumor samples reveals novel signatures of cancer. *Nucleic Acids Research*, 43(3), pp.1345-1356. doi: 10.1093/nar/gku1392 Disponible en: https://doi.org/10.1093/nar/gku1392
7. Biblioteca Rector Gabriel Ferreté, UPC (2016), Accessibilitat en obert dels aarticles UPC publicats el 2015. [online] Disponible en: https://upcommons.upc.edu/bitstream/handle/2117/90843/mescitatsUPC2015AO.pdf?sequence=6&isAllowed=y [Consulta: 10 Ene. 2020]
8. Clarivate Analytics (n.d.) *Web of Science*. [online] Disponible en: https://apps.webofknowledge.com/full_record.do?product=WOS&search_mode=GeneralSearch&qid=4&SID=F16Mb8Ye7BtCeZjdBkG&page=1&doc=1 [Consulta: 10 Ene 2020]
9. Oxford Academics (n.d.) *Crossref*. [online] Disponible en: https://academic.oup.com/nar/crossref-citedby/2411389 [Consulta: 10 Ene. 2020]
10. Google (n.d,) *Google Scholar* [online] Disponible en: https://scholar.google.com/scholar?q=link:https%3A%2F%2Facademic.oup.com%2Fnar%2Farticle%2F43%2F3%2F1345%2F2411389 [Consulta: 10 Ene. 2020]
11. Reixachs I Solé, M. (2015) *Identification of alternative splicing alterations in small cell lung cancer*. Disponible en: http://hdl.handle.net/10230/24981
12. Guo, W., Tzioutziou, N., Stephen, G., Milne, I., Calixto, C., Waugh, R., Brown, J. and Zhang, R. (2019). 3D RNA-seq - a powerful and flexible tool for rapid and accurate differential expression and alternative splicing analysis of RNA-seq data for biologists. doi: http://dx.doi.org/10.1101/656686
13. Biostars (n.d.) *Question: question about isoform switching searching tools*. [online] Disponible en: https://www.biostars.org/p/341956/ [Consulta: 10 Ene 2020]

CONSULTA GENERAL

14. Brown T. A. (2007). *Genomes 3*. 3rd ed. USA: Garland Science Pub.

15. Brown T. A. (2018). *Genomes 4*. 4th ed. USA: Garland Science Pub.
16. Witten, I. y Frank, E. (2005). *Data mining*. 2nd ed. San Francisco, Calif.: Morgan Kaufmann.
17. Jimeno A. y Ugedo l. (2008) Biologia 1r Batxillerat. Barcelona, España: Grup Promotor / Santillana Educación.